Reconocimiento no-supervisado de escenas mediante características extraídas de redes neuronales pre-entrenadas

TRABAJO DE FIN DE GRADO

UNIVERSIDAD AUTONOMA DE MADRID

Autor: Alejandro Gilabert Ramírez.
Tutor: Miguel Ángel García García

9 Julio 2020

Video Processing and Understanding Lab

Reconocimiento no-supervisado de escenas mediante características extraídas de redes neuronales pre-entrenadas

Autor: Alejandro Gilabert Ramírez.
Tutor: Miguel Ángel García García

Trabajo parcialmente financiado por el Ministerio de Economía y Competitividad del Gobierno de España bajo el proyecto TEC2017-88169-R (MobiNetVideo) (2018-2020)

“Si se cree y se trabaja, se puede“ Diego Pablo Simeone

Agradecimientos

Para empezar quería dar las gracias al departamento VPU de la EPS y sus integrantes por publicar gran número de trabajos de fin de grado relacionados con la inteligencia artificial que despertaron mucha curiosidad en mí y me convencieron a la hora de realizar este estudio.

En especial, quería agradecer a mi tutor, Miguel Angel, por la dedicación, implicación y referencia en momentos de dificultad que han sido de una indispensable ayuda para que se haya podido desarrollar este trabajo.

Más allá de esto, quiero agradecer a todos los profesores que me han impartido clase, ya que en mayor o menor medida, he sacado conclusiones, conocimeinto e información positiva de cada una de las asignaturas vistas.

Durante esta etapa de mi vida no sólo he visto conceptos nuevos acerca de las telecomunicaciones sino que he sido capaz de aprender a pensar por mi mismo, desarrollar un interés hacia temas relacionados con la ingeniería y el valor de las relaciones de compañerismo y amistad dentro de un campo tan técnico como es la ingeniería.

Mención especial tienen muchos de mis compañeros junto con los que he recorrido momentos de alegría, felicidad, frustración y desesperación, pero sobre todo hemos sido capaces de hacerlo juntos, conocedores de que en la unión se basa la fuerza y recibiendo un apoyo constante y recíproco entre nosotros.

Por último y no menos importante, quería agradecer a mis padres que me han facilitado enormemente el haber podido disfrutar y vivir esta esperiencia universitaria que sin lugar a dudas ha marcado un antes y un después en mi vida.

Resumen

Las redes neuronales han crecido de manera exponencial desde lo que podría considerarse el inicio de esta tecnología con el surgimiento de la idea del perceptrón y el perceptron multicapa a finales de 1950 e inicios de 1960.

A partir de ese momento, los avances continuaron, y con ellos se desarrollaron nuevas interpretaciones del algoritmo de "Backpropagation", una revolucionaria mejora que permitía calcular el error en la salida y propagarlo hacia capas anteriores y que nuestra red clasifique correctamente las imágenes. Más adelante surgieron las redes neuronales convolucionales que toman su inspiración en el cortex visual de los animales y con ella aparecieron los primeros estudios de reconocimiento de letras manuscritas, el considerado ”Hello World” en las redes neuronales.

A principios de siglo se aprovechó el poder de las GPU y los problemas computacionales desaparecieron y se superó una barrera que permite una capacidad de computo casi ilimitada a estas redes. Todos estos avances han propiciado que esta tecnología se encuentre muy presente en el día a día y se puede observar en sectores tan diferenciados como son la medicina, finanzas, automoción, marketing y un sin fin de ejemplos.

Este trabajo se centra en el reconocimiento de escenas basado en la extracción de características a partir de una red convolucional preentrenada. Para ello, se ha seguído el estudio de Andreas Sebastian Wolters *Unsupervised scene and place recognition based on features extracted from pretrained convolutional neural networks* en el que expone el uso de redes neuronales preentrenadas como VGG16 y AlexNet en primera instancia con un algoritmo de emparejamiento que almacene una tabla de lugares detectando si una escena es nueva o por el contrario ya ha sido encontrada una escena similar. Posteriormente se implementa un algoritmo de clustering k-means capaz de agrupar en distintas clases las imágenes a partir del diccionario de lugares.

En este proyecto se ha pre-entrenado las redes neuronales con la base de datos de imágenes de Imagenet lo que ha permitido un rápido entrenamiento sobre el dataset propio y la posibilidad de no utilizar una base de imágenes excesivamente grande para entrenar el modelo.

En última instancia, se ha realizado una comparación y análisis de los resultados obtenidos y comparación de los mismos en función de los diferentes métodos utilizados.

Palabras Clave

Red neuronal convolucional, reconocimiento de imágenes, reconocimiento de escenas, bases de datos, deep learning, clustering, clasificación

Abstract

Neuronal networks have been growing enormously from the moment that could be considered as the beginning of this technology with the "perceptron.ªnd the "multilayer perceptronïnvented in the late 50s and the beginning of the 60s.

From that moment on, improvements kept coming, and new approaches of the "backpropagation.ªlgorithm came out. This was an extraordinary revolution that made available returning back to previous layers and obtain the cost function and make small changes that lead to our network to classify correctly the images.

In the further years, improvements continued and with them appeared the convolutional neural network, based on the visual cortex of animals. The first papers related to hand-written figures recognition, the "hello World.ºf this field.

In the current century, the technology of GPUs was used and the problems related to computational maths disappeared with them what leads to an almost unlimited power of this networks.

All this discoveries and improvements have made this technology to be very present in our daily lives and can me mentioned in fields like medicine, financial markets, automotion and tons of other examples.

This paper will focus in place recognition based in feature extraction from a pre-trained neural network. For this, Andreas Sebastian Wolters' paper *Ünsupervised scene and place recognition based on features extracted from pre-trained convolutional neural networks"* has been followed, where CNNs as AlexNet or VGG16 are used firstly with a matching algorithm for the place recognition where a look-up table formed by different and new places is created and then a k-means clustering algorithm for the scene recognition based on the dictionary generated before.

For this project, pre-trained neural networks have been used with the ImageNet dataset that enabled a quick train and the possibility of not using a huge dataset of own images for training our model.

Finally, a comparison and analysis have been done taking into account the different measurements used and the results provided.

Key Words

Convolutional neural network, image recognition, scenes recognition, database, deep learning, clustering, classification.

Índice

Índice de figuras

1. Introducción

Las tecnologías basadas en la IA ya están siendo utilizadas para ayudar a los seres humanos a beneficiarse de mejoras y avances significativos.

Algunos ejemplos de las aplicaciones de la IA que están creciendo rápidamente en la actualidad y cuya demanda se ha multiplicado, pueden ser el procesamiento eficiente y escalable de datos, Distribución de contenido en las redes sociales, Reconocimiento de imágenes, clasificación y etiquetado y un sin fin de tecnologías más.[1].

Dentro de la inteligencia artificial uno de los enfoques principales es el aprendizaje automático, donde destaca el aprendizaje profundo que se utiliza para resolver problemas complejos que requieren grandes cantidades de datos. Este aprendizaje se produce mediante uso de redes neuronales que se organizan en capas para reconocer relaciones y patrones complejos en los datos [2], en este caso nos centraremos en la visión artificial, el campo que engloba este estudio y sobre el que se profundizará de todos los aspectos que hayan tenido relación durante la realización del mismo.

1.1. Motivación

Las redes neuronales convolucionales (CNN) [3] tienen hoy en día gran cantidad de aplicaciones y utilidades que sin ninguna duda continuarán creciendo de manera exponencial en el futuro. Tanto por el presente como por el futuro que se le reserva a estas tecnologías, se convierten en un campo digno de estudio y de conocimiento de sus fundamentos y aplicaciones más básicas.

Como se sugiere en ciertos estudios [4], los rendimientos de estas CNN están altamente relacionados con la cantidad de datos disponibles. A pesar de ello, se ha visto en diferentes trabajos [5] cómo con una cantidad pequeña de capas se puede transferir este conocimiento para otras áreas de desarrollo. Estos resultados demuestran cómo una red neuronal convolucional entrenada sobre un gran dataset es capaz de aprender información de características de las imágenes y transferirlas a otros dataset que no sean los originales.

Una de las motivaciones principales de este trabajo ha sido además de conocer esta tecnología en mayor profundidad, el poder encontrar gran cantidad de documentación y código disponible de manera open source. Otra de las motivaciones de cara a este estudio ha sido la posibilidad de realizarse el mismo usando el lenguaje de programación Python, sobre el cual se tenían unos conocimientos muy limitados antes de elegir este trabajo. Así mismo, poder autogestionar de manera didacta un proyecto de índole tecnológica hace verdaderamente atractiva la elección de este TFG.

1.2. Objetivos

Uno de los objetivos principales de este trabajo es el aprendizaje y entendimiento de las posibilidades que ofrece programar utilizando tensores y la potencia de la GPU para realizar operaciones y algoritmos de forma vectorial en lugar de la tradicional programación secuencial.

Más allá de esto, otro de los objetivos de este proyecto es la introducción al aprendizaje profundo y las posibilidades que estas ofrecen enfocadas hacia el tratamiento de imágenes. Se han utilizado las redes neuronales convolucionales pre-entrenadas AlexNet y VGG16, que permiten un enfoque ligeramente distinto debido a la estructura de estas redes.

En primera instancia, el objetivo era enfocar este estudio de clasificación y reconocimiento de escenas sobre distintas muestras tomadas del campus de la Universidad Autónoma de Madrid y las distintas facultades que forman la misma analizando las características obtenidas a través del algoritmo de extracción de características de los dataset propios creados con imágenes del campus y cuyo objetivo es reconocer el contenido de la escena además de si ese lugar ha sido analizado previamente o si por el contrario se trata de uno nuevo.

Debido a la situación adversa que ha provocado el COVID-19 desde mediados del mes de Marzo, hubo que darle otro enfoque a este trabajo, dónde se decidió cambiar el tipo de imágenes a utilizar conservando la idea principal del estudio. Para poder solventar los problemas del confinamiento y la imposibilidad de tomar fotografías de los edificios de la UAM, se decidió dividir el estudio en dos bloques diferentes.

El primero de ellos se encargaría de diferenciar 2 clases muy distintas la una de la otra. Para ello, se ha utilizado un dataset formado por imágenes del exterior y otro dataset formado por imágenes del interior de un domicilio. El segundo bloque estaría formado por un conjunto de imágenes del interior de un casa y se busca una clasificación de las distintas dependencias que conforman un domicilio.

1.3. Organización

De cara a la organización de este trabajo, en primer lugar se hace una exposición al estado del arte que envuelve una descripción técnica y funcional de las herramientas que se han utilizado directa o indirectamente. Más adelante, se explica el proceso que se ha llevado a cabo: estudio, desarrollo, integración, pruebas realizadas y los resultados obtenidos. Para finalizar, se procede a hacer una reflexión sobre la realización de este estudio además de posibles estudios posteriores.

2. Estado del arte

2.1. Introducción

En esta parte del trabajo se van a explicar las diferentes tecnologías que se han tenido en cuenta para la realización del proyecto. Para ello, se ha realizado un análisis técnico y fundamental que ponga en manifiesto el conocimiento de estas tecnologías.

2.2. Python

El lenguaje utilizado para este trabajo ha sido Python, un lenguaje de programación interpretado, orientado a objetos de alto nivel. Este lenguaje hace énfasis en la legitimidad del codigo lo que favorece una fácil depuración.

Python fue creado como lenguaje de programación de uso general aunque dispone de una librerías y entornos de desarrollo específicos para Data Science [6]. Esto, añadido a su potencia, su disponibilidad (open source) y su facilidad de aprendizaje le ha llevado a tomar la delantera a otros lenguajes como pueden ser SAS, uno de los lideres comerciales [7] y R (también open source, pero más propio de entornos académicos o de investigación).

La cantidad de librerías que contiene, tipos de datos, soporte en programación orientada a objetos y funciones incorporadas al lenguaje además de ser gratuito lo convierten en una de los principales motivos por los que se ha decidido usar este lenguaje.

Para el desarrollo, se está utilizando Anaconda Distribution, un conjunto Open Source de aplicaciones, librerías y conceptos para el desarrollo de Data science en Python [8]. Uno de los paquetes que más se ha utilizado es el de Conda, un entorno de gestión de entornos y dependencias disponible para casi cualquier lenguaje además de compatible para sistemas Windows, Linux and MAcOS [9].

2.3. Graphic Processing Unit

Las unidades de procesamiento gráfico (GPU) [10] se basan en un microprocesador suplementario de la tarjeta gráfica dedicado al procesamiento de señales, operaciones aritméticas y demás operaciones para reducir la carga de trabajo del procesador principal que puede utilizar la CPU para otros cálculos [10].

Una de las ventajas más visibles que podemos encontrar al uso de la GPU es la diferencia de velocidad entre la CPU y la propia GPU, la cual es mucho más eficiente si los datos son lo suficientemente grandes, ya que estas están diseñadas para realizar una única tarea especifica [11]. En este trabajo ha sido imprescindible el uso de la GPU ya que las diferencias que encontramos de rendimiento hacen que únicamente sea viable para equipos que dispongan de una tarjeta gráfica dedicada. Para la realización de este proyecto se ha utilizado un PC de los ordenadores del laboratorio VPU de la Escuela Politécnica Superior, los cuales disponen del modelo "NVIDIA GeForce GTX 8800GTS."

2.4. CUDA

La compañía americana Nvidia, especializada en el desarrollo de unidades de procesamiento gráfico, en el año 2007 crearon una plataforma de computación llamada CUDA (Compute Unified Device Architecture) [12].Esta está compuesta distintos módulos que actúan como pequeños procesadores encargados de realizar una tarea de manera paralela, con múltiples hilos operaciones independientes

Una de las grandes revoluciones de estos núcleos es su capacidad para procesar códigos C, C++ y Fortran directamente a la GPU sin utilizar un código de ensamblaje. Con esta simplificación aprovecha la computación paralela en la que se ejecutan simultaneamente miles de tareas o subprocesos. [13].

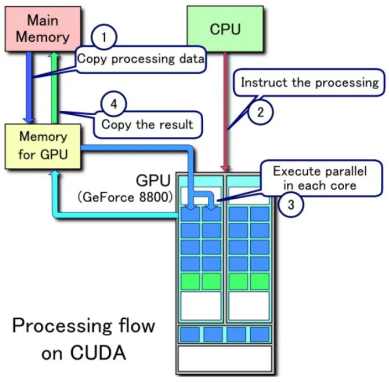


Figura 1: Funcionamiento de una tarjeta gráfica NVIDIA GeForce

2.5. Pytorch

Dentro del lenguaje Python, existen gran cantidad de librerías que tienen potencial suficiente para que se utilicen en campos de la inteligencia artificial y aunque todavía es un lenguaje joven, Pytorch está siendo capaz de hacerle la competencía a TensorFlow, otro framework desarrollado por Google [14]. Pytorch es un paquete de computación científica que se basa en el uso de los procesadores gráficos [15]. Además del cálculo de tensores gracias a la ayuda de la GPU, permite la construcción de redes neuronales profundas a partir de sistemas de almacenado de operaciones [16].

Alguna de las muchas ventajas que ofrece Pytorch es que se pueden seguir usando paquetes como "NumPy", "SciPy", tiene una API muy sencilla e intuitiva, nos aporta la facilidad de programarlo completamente en Python y la existencia de una comunidad cada vez más grande en la que se pueden encontrar gran cantidad de estudios y de material para el desarrollo de nuestra red neuronal.

2.6. Keras

Keras es un framework de alto nivel para el aprendizaje, escrito en Python. Fue desarrollado con el objeto de facilitar un proceso de experimentación rápida. Esta es una de sus grandes ventajas en comparación con otros frameworks destinados al estudio de la inteligencia artificial. El modelo de la red neuronal VGG16, la cual será explicada más adelante y que se ha utilizado en este estudio, es compatible con Keras, lo cual nos ayuda de manera exponencial gracias a funciones intuitivas.

2.7. Google Collab

Google Collab es un entorno de programación que permite ejecutar y programar en Python sin la necesidad de una instalación previa ni una configuración del propio entorno. Aparte de ello, tiene acceso gratuito a GPU, algo esencial para la realización de este proyecto. Otra de las características imprescindibles es que nos permite instalar y descargar sub-librerías simplemente introduciendo un comando como si de la terminal se tratara.

Otro factor diferencial es la codificación por bloques, lo que posibilita una fácil depuración y detección de posibles errores. Por todo ello, se ha decidido utilizar este entorno en detrimento de la cuenta de los laboratorios de VPU.

2.8. Redes Neuronales Convolucionales

Las CNN (Convolutional Neural Network) son un tipo de Red Neuronal Artificial que procesa sus capas imitando al cortex visual del ojo humano para identificar distintas características en las entradas que en definitiva hacen que pueda identificar objetos. Para ser capaz de esto, la red contiene varias capas ocultas especializadas y jerarquizadas. Esto quiere decir que las primeras capas pueden detectar lineas, curvas y se van especializando hasta llegar a capas más profundas que reconocen formas complejas.

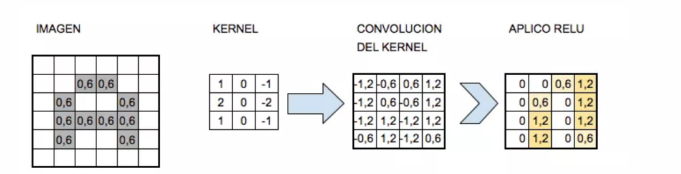


Figura 2: Convolución de imagen con Kernel y aplicación de función de activación.

En primer lugar, se aplica un pre-procesamiento de la imagen, que habitualmente se basa en normalizar los valores de entrada para obtener valores entre 0 y 1. Posteriormente, se aplica una convolución. Esto consiste en seleccionar un grupo cercano de pixeles de la imagen e ir operando en función de una matriz llamada kernel. Estos kernels, que a su vez se llaman filtros en su conjunto, recorren todas las neuronas de entrada y generan una matriz de salida, la cual será la nueva capa de neuronas ocultas.

Una vez se haya realizado la convolución con el kernel, se procede a utilizar la función de activación. Durante mucho tiempo, se han utilizado las funciones no lineales sigmoide y tangente hiperbólica. El problema de estas funciones es el límite de sensibilidad y saturación, el cual puede resultar un problema para el algoritmo y su capacidad para determinar los pesos y así mejorar el rendimiento de nuestro modelo [17]. A continuación se pueden observar algunos de las distintas funciones de activación que se pueden usar.

Problem Type	Output Type	Final Activation Function	Loss Function
Regression	Numerical value	Linear	Mean Squared Error (MSE)
Classification	Binary outcome	Sigmoid	Binary Cross Entropy
Classification	Single label, multiple classes	Softmax	Cross Entropy
Classification	Multiple labels, multiple classes	Sigmoid	Binary Cross Entropy

Figura 3: Funciones de coste para los problemas más comunes [18]

Por norma general, es común que a la hora de añadir capas nuevas, si la última capa de nuestro modelo utiliza un clasificador, se use una función de activación que devuelva un vector con una distribución de probabilidad comprendida entre 0 y 1 [19].

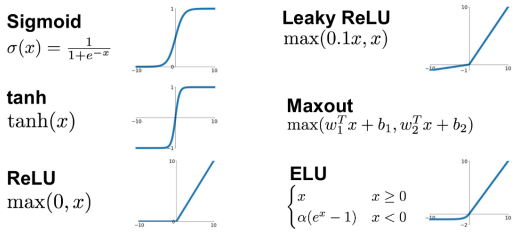
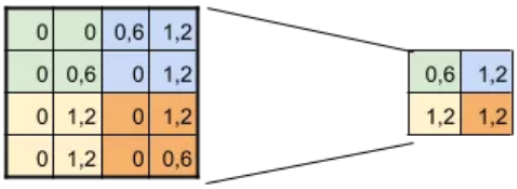


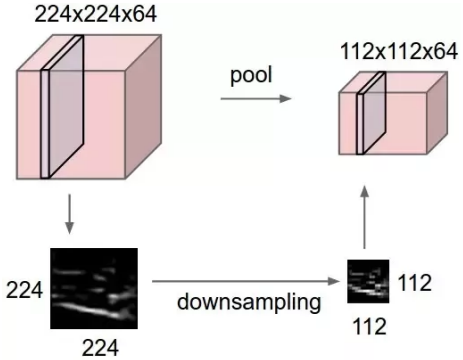
Figura 4: Principales funciones de activación

Una vez aplicada la función de activación llega el punto en el que se reduce la cantidad de neuronas antes de realizar una nueva convolución. ¿Por qué de esta reducción? A partir de nuestra primera capa y teniendo en cuenta que se trata de una imagen en formato RGB, el cual consta de 3 canales, [20] el número de neuronas podría llegar a ser demasiado alto y esto implica un alto procesamiento de datos el cual podría resultar perjudicial para los tiempos de ejecución de nuestro modelo.

Una de las soluciones planteadas para reducir el tamaño de la siguiente capa será un submuestreo que reduzca el tamaño de las imágenes haciendo prevalecer las características más importantes de cada filtro. En los modelos pre-entrenados que se han utilizado se ha podido observar cómo este submuestro se realiza a través de "Max Pooling".



(a) Representación Pictorial 2x2



(b) Ejemplo real

Figura 5: Max-pooling

Como bien se puede observar en la imagen anterior, el principal objetivo de este tipo de muestreo es recorrer nuestra imagen de izquierda a derecha y de arriba a abajo, pero en vez de tomar uno a

uno los pixeles, solamente se preservará el valor más alto. Con esta operación no solo se consigue reducir notablemente el número de neuronas, sino que a su vez, se sigue conservando la información más importante de cara a detectar las características deseadas.

De vuelta en el modelo, la primera convolución es capaz de detectar características primitivas como lineas ó curvas. A medida que se hagan más capas con las convoluciones, los mapas de característi-casserán capaces de reconocer formas más complejas. Tras realizar sucesivas convoluciones donde se ha-yan extraido las características necesarias, que quedarán almacenadas en un mapa de características.Una vez estas hayan sido extraidas, se tendrán una o varias capas completamente conectadas además de una capa Softmax que configura una función de activación [21].

Las capas completamente conectadas son aquellas que cogen los resultados de una convolución/pooling y lo utilizan para clasificar la imagen en una etiqueta. La salida de la convolución se representa en un solo vector de valores los cuales representan una probabilidad de que cierta característica pertenezca a una etiqueta. [22].

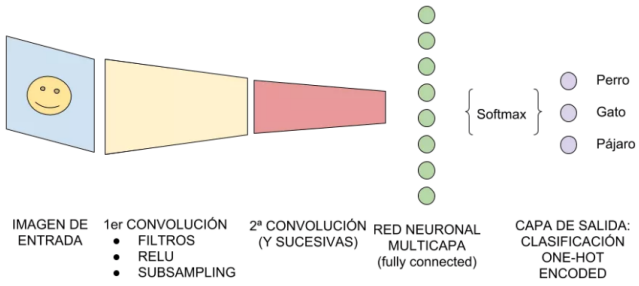


Figura 6: Esquema Red neuronal convolucional

La función SoftMax es la función de activación de la salida después de la última capa completamente conectada. En la salida del clasificador, la función asocia cada una de las entradas a una de las diferentes clases. La salida de la función SoftMax está comprendida entre 0 y 1, por esta razón, es una función adecuada para la representación de funciones de probabilidad.

Descenso por gradiente

Una vez que se ha determinado la arquitectura de nuestra red neuronal, es importante conocer cómo es capaz de tratar el modelo el input que se quiere que reconozca. En este punto entra en escena el mencionado descenso por gradiente.

Para explicar este concepto es bueno recordar el funcionamiento de una red neuronal, donde las redes se componen de neuronas que se organizan en capas, donde menos en la capa de entrada, cada neurona es un sumatorio de todas las entradas, que no son más que las salidas de las capas anteriores multiplicadas por unos pesos. A esta suma se añade un término adicional llamado sesgo o bias y posteriormente se aplica la ya explicada función no lineal de activación. [23].

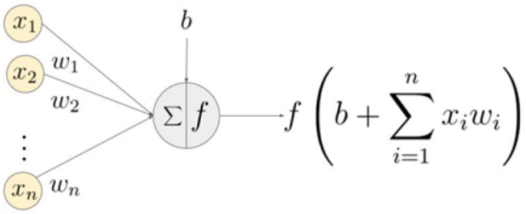


Figura 7: Neurona con entrada, pesos correspondientes, bias y la función de activación f

Estos parámetros de la red (pesos y bias) son valores numéricos que se trata de ajustar mediante el entrenamiento. Si se parte sin un modelo pre-entrenado es muy común inicializar los pesos aleatoriamente y los bias a cero. Con esto se consigue evitar que todas las neuronas acaben aprendiendo lo mismo. A partir de aquí, se procede a iterar siguiendo un algoritmo de optimización que busque minimizar la diferencia entre la salida real y la estimada por nuestro modelo.

El algoritmo de optimización usado para el entrenamiento de redes neuronales es el descenso por gradiente. Es un cálculo que nos permite ajustar los parametros de la red sin que se minimice la desviación a la salida. En este trabajo se ha utilizado un descenso por gradiente en mini batch, donde se alimenta a la red con N muestras en cada iteración. Con ello conseguimos un entrenamiento más rápido gracias a la paralelización de las operaciones.

En cuanto a las iteraciones del algoritmo de descenso por gradiente en primer lugar se introduce un sistema de lotes conocido como batch con N muestras aleatorias provenientes del dataset de entrenamiento. A continuación, se obtienen como resultado las predicciones a la salida después de los cálculos pertinentes en cada capa del modelo, esta etapa es conocida como forward propagation, donde se avanza desde la entrada a la salida en una dirección [24].

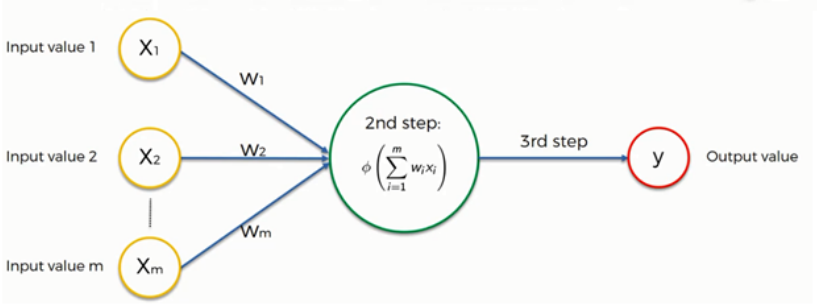


Figura 8: Esquema de la propagación hacia adelante [25]

Una vez hecha la propagación hacia adelante, se evalúa la función de coste (también llamada función de pérdida) para dicho lote. El valor de esta función de coste es lo que se trata de minimizar en todo momento mediante el algoritmo, y hacia ello se orientan los siguientes pasos:

Se calcula el gradiente como la derivada de la función de coste con respecto a todos los parámetros de la red. Gráficamente sería la pendiente de la tangente a la función de coste en el punto donde uno se encuentra (evaluando los pesos actuales). Desde un punto de vista matemático es un vector que proporciona la dirección y el sentido en que dicha función aumenta más rápido. La idea por tanto es movernos en sentido contrario al del gradiente, tratando de alcanzar el mínimo global [26].

El cálculo del vector gradiente en una red neuronal profunda no es para nada trivial, se complica bastante debido a la gran cantidad de parámetros y a su disposición en múltiples capas. Una de las

preguntas que uno debe plantearse es saber cuánto y cómo influye la variación de un parámetro de la primera capa en el coste final si esa variación repercute en todas las neuronas de todas las capas sucesivas. Esto es posible saberlo gracias al algoritmo de back-propagation.

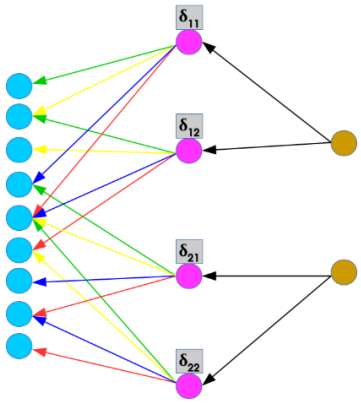


Figura 9: Esquema de gradientes generados según la propagación hacia atrás [27]

Este algoritmo se caracteriza por su búsqueda de la disminución del error en las distintas capas que componen nuestro modelo. La forma de calcular la importancia que tiene cada neurona en el error es lo que da sentido al nombre de backpropagation, ya que en primera instancia se calcula la çulpa"del error de cada neurona de la última capa y lo va propagando hacia atrás. Se podría decir que pondera el reparto del error para cada una de las neuronas de la red. El algoritmo de propagación hacia atrás determina las derivadas parciales de la función de coste con respecto a cada una de las variables [28].

Una vez obtenido el vector gradiente, se actualizan los parámetros de la red restando a su valor actual, el valor del gradiente correspondiente multiplicado por una tasa de aprendizaje que permite ajustar la magnitud de nuestros pasos.

El término de actualización se resta porque se quiere avanzar en el sentido contrario al del gradiente para que la función de coste disminuya. Cuanto más cerca se esté del mínimo global, los pasos serán en teoría más pequeños porque la pendiente de la función de coste será menor y se suele optar por ir disminuyendo la tasa de aprendizaje con el tiempo. Se repiten todos los pasos mientras el valor de la función de coste y las métricas de salida no empiecen a empeorar de forma sostenida.

Una vez hecho esto, se prosigue con el objetivo de encontrar el mínimo global de la función de coste y no acabar rebotando en un mínimo local o un punto de silla (mínimo local de una dimensión y máximo local de otra).

De cara a evitar un final no deseado será importante la elección de la tasa de aprendizaje y por consiguiente el valor que tenga, que no debe ser ni muy alto ya que puede producir oscilaciones. Tampoco debe ser muy bajo, lo que produciría una mayor lentitud y número de iteraciones.

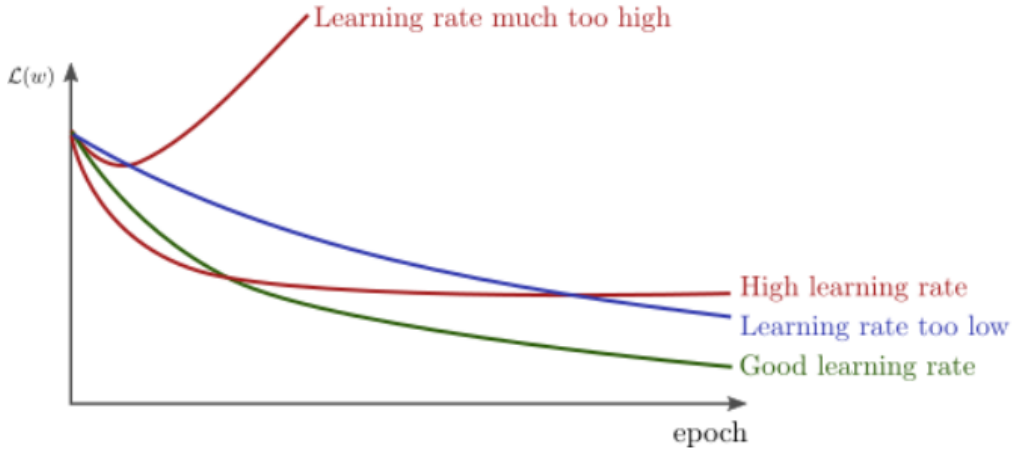


Figura 10: Tasa de aprendizaje en función de las épocas [29].

Con medir el valor no es suficiente ya que hay que tener en cuenta si la tasa es variable o fija en el tiempo, donde se puede hacer que vaya disminuyendo con el tiempo para evitar oscilaciones cuando se está llegando a un mínimo de la función de coste, o si será la misma para toda la red. En nuestro caso, al utilizar una transferencia de Aprendizaje con las redes pre-entrenadas AlexNet y VGG16, conviene elegir una tasa baja para entrenar la para la parte del modelo pre-entrenado, y otra más alta para las capas que se introduzcan.

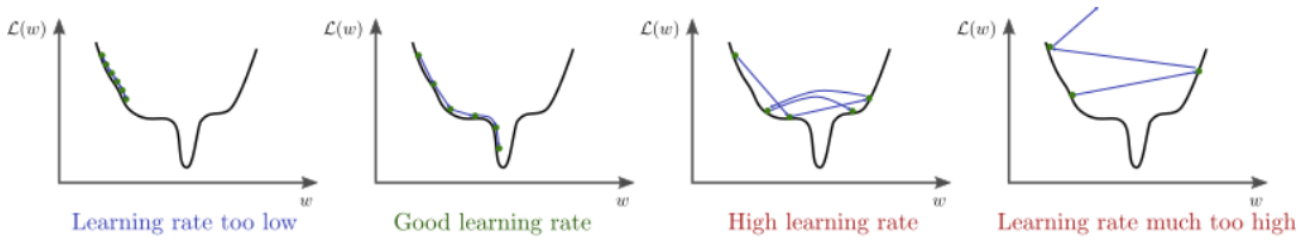


Figura 11: Diferentes iteraciones en las tasas de aprendizaje [29].

Existe la posibilidad de que nuestro modelo no sea capaz de aprender tal cantidad de parametros sin producir sobreajuste. Esto sucede cuando el modelo se comporta bien en las pruebas de entrenamiento pero se convierte difícil para el modelo reconocer nuevos ejemplos que no se encuentran en nuestro trainingset. Una de las distintas posibilidades es que si tenemos sobreajuste se debe a que el modelo reconoce imágenes específicas en vez de patrones generales [36]. Para ello, se utilizan las siguientes técnicas para combatir el sobreajuste.

2.8.1. Data Augmentation

La manera más sencilla y común para reducir el sobreajuste en los datos de una imagen es artificialmente alargar el dataset utilizando transformaciones sobre las imágenes iniciales. Se emplean generalmente dos formas distintas las cuales permiten obtener imágenes transformadas a partir de las originales con un coste computacional muy bajo y sin necesidad de guardar en disco las imágenes.

Una de las posibles técnicas sería el uso de "Data augmentation por espejo". Con este método se consigue un dataset del doble de tamaño simplemente rotando sobre el eje vertical una imagen como se puede apreciar en la imagen a continuación.

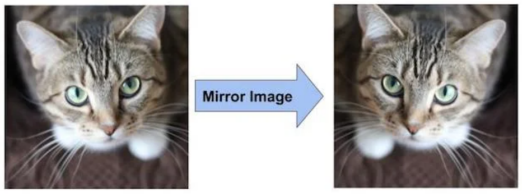


Figura 12: Data Augmentation por espejo

Otra posible solución para reducir el sobreajuste es el uso de cortes aleatorios. En la red AlexNet extrageron cortes aleatorios de tamaño 227x227 de las imágenes de 256x256 para utilizarlos como entradas de la red. Con ello se aumentó el tamaño de los datos por un factor de 2048.

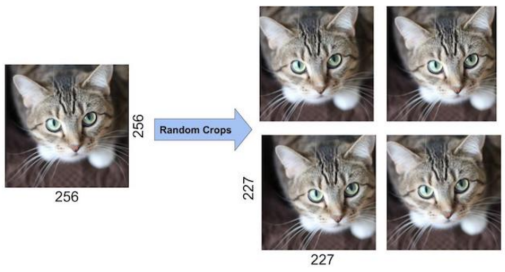


Figura 13: Data Augmentation con centrado aleatorio

Se puede observar cómo las cuatro imágenes parecen muy similares pero no son exactamente la misma. Esto enseña a nuestra red que un pequeño desplazamiento de los píxeles no cambia el hecho de que la imagen siga teniendo el mismo contenido. Si data augmentation hubiera sido imposible el utilizar un dataset tan largo porque habría sufrido de sobreajuste.

Se puede observar cómo las imágenes generadas son muy similares entre ellas aunque con ligeras diferencias. Con ello se pretende enseñar a nuestro modelo que un pequeño desplazamiento de los pixeles no cambia el contenido de una imagen. Sin esta técnica hubiera sido imposible usar un dataset tan grande porque se hubiera sufrido sobreajuste.

2.8.2. Dropout

Combinar diferentes imágenes para aumentar el tamaño del dataset es una opción muy útil para reducir los errores en el entrenamiento, pero para redes neuronales muy grandes que tarden días en entrenarse tiene un coste alto. Por ello se introdujo la técnica llamada "dropout". Esta técnica la introdujo G.E. Hinton en otro estudio en 2012 [30].

El método Dropout consiste en poner a cero la salida de cada neurona oculta con probabilidad 0.5. Por tanto, las neuronas que han sido omitidas no contribuyen a la propagación hacia adelante ni hacia atrás. Entonces, cada vez que se presenta una entrada, la red muestra una arquitectura diferente aunque compartan los mismos pesos. Con esta técnica se consigue reducir las complejas

adaptaciones de las neuronas ya que estas aprenden características más robustas que son útiles junto con muchos subconjuntos aleatorios diferentes de otras neuronas.

Durante el test no se realiza este proceso y toda la red es utilizada, pero la salida es escalada por un factor de 0.5 debido a las la fase de entrenamiento en la que no se han usado estas neuronas.

Antiguamente, las GPU utilizaban alrededor de 3 GB de memoria. Esto era especialmente problemático porque el set de entrenamiento tenía más de 1 millón de imágenes. Se permite entrenamiento multi-GPU usando la mitad de las neuronas de nuestro modelo en una GPU y la otra parte en otra GPU. Esto permite entrenar un modelo más grande además de reducir los tiempos de entrenamiento.

2.9. Transfer Learning

Los seres humanos tienen la habilidad inherente de transferir el conocimiento entre diferentes tareas. Cuanto más relacionadas están estas tareas, más sencillo será resolver estos problemas de reconocimiento.
Convencionalmente, los algoritmos de deep learning han sido diseñados para trabajar de manera aislada, ya que han sido entrenados para realizar una tarea específica. La idea sobre la que se basa la transferencia de conocimiento consiste en superar el paradigma del aprendizaje aislado y utilizar el conocimiento adquirido para resolver una tarea similar de forma que se puedan redefinir tanto características como pesos así como resolver problemas de datos utilizando datasets más pequeños. [31].

Transfer learnign es un método popular en data science debido a que permite crear modelos precisos en un tiempo mucho menor. Con ello, en vez de empezar el proceso desde 0, se tienen unas bases que se han aprendido resolviendo un determinado problema. El transfer learning es muchas veces expresado como el uso de modelos pre-entrenados. Un modelo pre-entrenado es un modelo que fue entrenado en un dataset suficientemente grande como para resolver no tener que perder ese tiempo entrenando la red cuando el dataset utilizado cumple los requisitos del estudio. De acuerdo con el coste de entrenamiento de estos modelos es muy común importar y usar estos modelos.

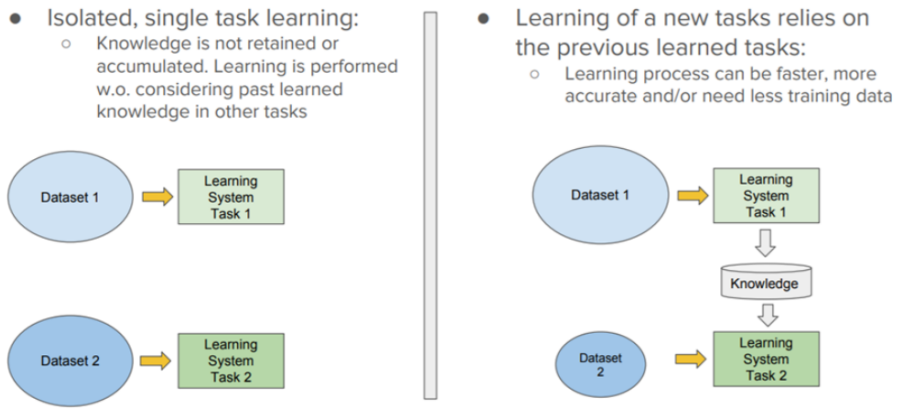


Figura 14: Aprendizaje tradicional vs Transfer Learning

Cuando se utiliza un modelo pre-entrenado para resolver un objetivo propio se empieza por quitar el clasificador original, añadir uno nuevo que se adapte a nuestras necesidades y finalmente tendremos que hacer fine-tune a nuestro modelo según 3 estrategias:

1) Entrenar el modelo entero. En este caso se utiliza la arquitectura de la red pre-entrenada y se entrena de acuerdo a tu dataset. Se está aprendiendo desde el principio por lo que será necesario un dataset grande y mucho poder computacional.

2) Entrenar algunas capas y dejar las demás congeladas. Como se ha comentado anteriormente, las primeras capas se refieren a características generales mientras que las capas altas se refieren a caracteristicas específicas. Aquí se valora la cantidad que se quieren ajustar los pesos de la red. Normalmente, si se tiene un dataset pequeño y un número largo de parámetros se suelen dejar unas cuantas de estas capas congeladas para evitar sobreajuste. Por el contrario, si el dataset es largo y el número de parámetros pequeño, se puede mejorar el modelo entrenando más capas para el nuevo objetivo mientras el sobreajuste no sea un problema.

3) Congelar la parte convolucional. Este caso corresponde a una situación extrema. La idea principal de esta medida es conservar la base convolucional en su forma original y utilizar sus salidas para alimentar al clasificador. Se está usando un modelo pre-entrenado como un mecanismo de extraccion de características lo que puede ser muy útil si se tiene un dataset pequeño o si el problema del modelo pre-entrenado está relacionado con el problema que se quiere solucionar [32].

A diferencia de la última posibilidad, las 2 primeras medidas tienen que tener mucho cuidado con la tasa de aprendizaje utilizada en la parte convolucional. Esta tasa se encarga de de controlar cuando se ajustan los pesos a la red. Cuando se está utilizando un modelo pre-entrenado basado en CNN es importante utilizar una tasa de aprendizaje pequeña porque las tasas de aprendizaje altas aumentan el riesgo de de perder el conocimiento previo, por lo que mantener una tasa de aprendizaje baja asegura que no se distorsionen los pesos en gran medida y demasiado pronto.

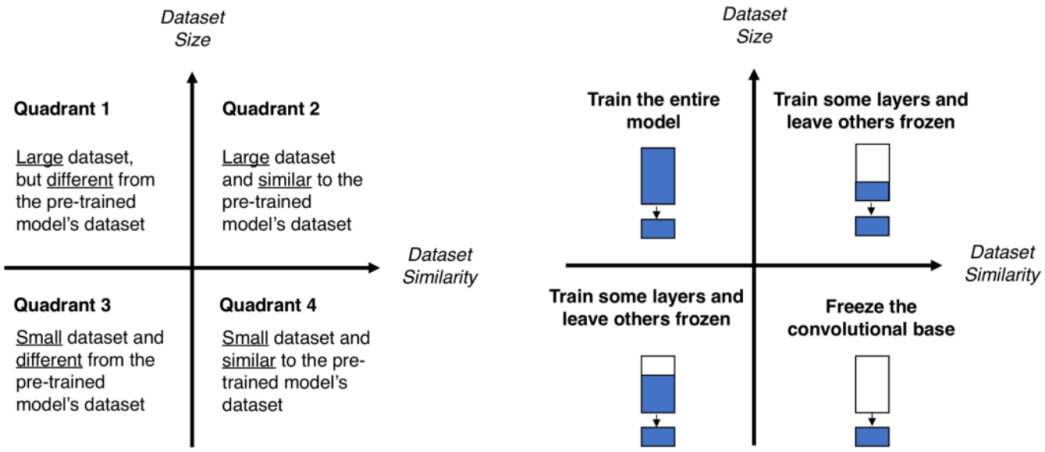


Figura 15: Matriz de similitud (izq) mapa de decisión.

Según la imagen se puede usar la matriz de similitud para elegir el modelo que más beneficiaría a nuestro objetivo.

Cuadrante 1: Dataset grande pero distinto del utilizado para el entrenamiento de la red. Si este es el caso, se podría utilizar la estrategia 1. Desde que se tiene un gran dataset se podrá entrenar al modelo desde 0 y hacer lo qie se quiera. En función de la no similitud de los dataset puede ser útil el inicializar el modelo pre-entrenado usando su arquitectura y pesos.

Cuadrante 2: Se utiliza un dataset grande relacionado con el usado por el modelo pre-entrenado. Para este caso cualquiera de las posibilidades sería valida, aunque la más eficiente sería la estrategia 2, teniendo en cuenta que tenemos un dataset grande, no resultarían problemas de sobreajuste. A pesar de ello, se podría ahorrar mucho esfuerzo en el entrenamiento si los dataset son similares. Asimismo, sería suficiente con entrenar el clasificador y las capas finales convolucionales.

Cuadrante 3: Para este caso se utiliza un dataset pequeño y con pocas similitudes con el del modelo pre-entrenado. Aparentemente estamos ante una situación complicada ya que se tendría que tener un balance muy compensado del número de capas que entrenar y cuales congelar. Si se modifica demasiado profundo el modelo se puede producir sobreajuste y por el contrario se correría el riesgo de que el modelo no estuviera aprendiendo nada útil.

Cuadrante 4: Se utiliza un dataset pequeño pero similar al del modelo pre-entrenado. Lo más sencillo sería eliminar la última capa conectada, entrenar el modelo como un extractor de características y usar estas mismas para entrenar un nuevo clasificador.

3. Diseño

3.1. Introduccion

Este capítulo recoge una de las fases más importante de este trabajo ya que se explicará el diseño utilizado para el desarrollo de este trabajo.

El objetivo de este TFG es la implementación y evaluación de un algoritmo de reconocimiento de escenas basado en la aplicación de técnicas de agrupación (clustering) no supervisada a conjuntos de características extraídas de redes neuronales convolucionales pre-entrenadas. Se irán exponiendo las diferentes configuraciones y tecnologías que se han utilizado para cumplir los objetivos marcados.

Los algoritmos de reconocimiento basados en redes neuronales profundas son mayoritariamente supervisados, siendo necesario disponer de conjuntos de entrenamiento con millones de imágenes etiquetadas. Esto dificulta la aplicación de estas redes en ámbitos donde no se dispone de bases de datos de imágenes de semejante envergadura.

En primer lugar se analizarán los dos modelos que se utilizarán en el desarrollo, AlexNet y VGG16. Se hará un análisis de su estructura y se hará hincapié en las capas convolucionales que nos interesan para el correcto funcionamiento del algoritmo de extracción de características y el reconocimiento de lugares y escenas.

En el último apéndice de este estudio, se ha añadido el código implementado de las funciones que se han desarrollado y se explicarán su uso de manera funcional en este capítulo.

3.2. AlexNet

AlexNet es una de las dos CNN que se han utilizado en el trabajo. Fue desarrollada en 2012 ganando la competición ImageNet LSVRC. AlexNet es una arquitectura de una red neuronal convolucional que resuelve el problema de clasificación de imágenes con una entrada de más de 1000 clases y un vector de salida de 1000 parámetros. Este vector de salida incluye la probabilidad de que una de las entradas corresponda a una de las clases de nuestra red. La suma de todas las probabilidades tiene que ser 1.

La entrada a AlexNet es una imagen en formato RGB de dimensiones 227x227. Esto significa que todas las imágenes tanto de entrenamiento como de test y validación deben tener esas dimensiones. Si la imagen no cumple estas condiciones, la dimensión de menor medida es convertida a 227 y el resultado es rescalado para obtener una imagen de 256x256. De la misma forma, si la imagen está en escala de grises, esta se convierte a RGB replicando la imagen para obtener los 3 canales de este formato[33].

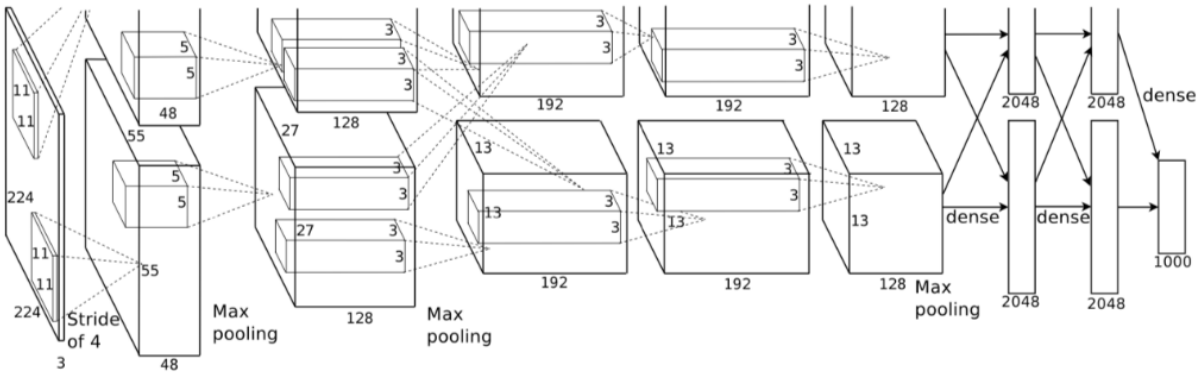


Figura 16: Arquitectura CNN AlexNet[34]

La red está compuesta por 8 capas con diferentes pesos. Las primeras 5 capas son capas convolucionales mientras que las restantes se tratan de capas completamente conectadas. La salida de la última capa alimenta a un clasificador con 1000 etiquetas de distintas clases [35]. La primera capa convolucional filtra la imagen inicial de dimensiones 224x224x3 con 96 kernels de tamaño 11x11x3 con un stride de 4 pixels. La segunda capa coge la entrada después de aplicar una normalización y un proceso de pooling y se filtra con 256 kernels de tamaño 5x5x48. Las 3 capas convolucionales siguientes están conectadas una a la otra sin aplicar ningúna capa de pooling o de normalización. Las capas completamente conectadas tienen 4096 neuronas cada una [36].

En la tabla que se muestra al final de este apartado, se pueden observar las diferentes capas, parámetros y unidades computacionales necesarias. En esta tabla se describen cada una de las operaciones.

Una caracteristica importante de AlexNet es el uso de ReLU (Rectified Linear Unit) no linear. La función sigmoide solía ser usarse de manera común para entrenar modelos de redes neuronales. AlexNet demostró que con el uso de ReLU no linear las CNN podían ser entrenadas mucho más rápido que usando una función sigmoide o tanh.

La ReLU no linear es aplicada después de todas las capas convolucionales y fully connected. Las dos primeras ReLU están seguidas de una normalización justo antes del pooling.

La imagen a continuación muestra como el estudio de Krizhevsky, autor original del trabajo AlexNet, aseguraba que con una ReLU se conseguía un 25 por ciento del ratio de error de entrenamiento 6 veces más rápido que utilizando tanh. Esto fue probado en el dataset de imágenes CIFAR-10.

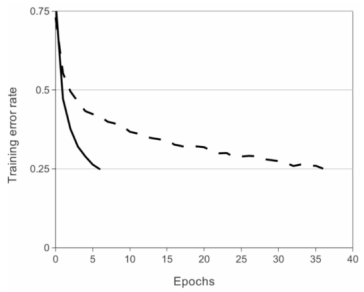


Figura 17: Mejora gracias al uso de ReLU [34]

Las capas de pooling normalmente son usadas para muestrear el alto y ancho de los tensores dejando constante la profundidad. Las capas de overlapping max pooling son muy parecidas a las de max pooling con la diferencia que las ventanas adyacentes sobre las que se calcula el máximo se superponen entre sí.

La arquitectura de la red tiene más de 60 millones de parámetros. Debido a esto, existe la posibilidad que nuestro modelo no sea capaz de aprender tal cantidad de parámetros sin producir sobreajuste. Esto sucede cuando el modelo se comporta bien en las pruebas de entrenamiento pero se convierte difícil para el modelo reconocer nuevos ejemplos que no se encuentran en nuestro training set. Para combatir esto se utilizan las técnicas de Dropout y data augmentation que se han visto anteriormente en el capítulo del estado del arte.

Size / Operation	Filter	Depth	Stride	Padding	Number of Parameters	Forward Computation
3* 227 * 227						
Conv1 + Relu	11 * 11	96	4		(11*11*3 + 1) * 96=34944	(11*11*3 + 1) * 96 * 55 * 55=105705600
96 * 55 * 55						
Max Pooling	3 * 3		2			
96 * 27 * 27						
Norm						
Conv2 + Relu	5 * 5	256	1	2	(5 * 5 * 96 + 1) * 256=614656	(5 * 5 * 96 + 1) * 256 * 27 * 27=448084224
256 * 27 * 27						
Max Pooling	3 * 3		2			
256 * 13 * 13						
Norm						
Conv3 + Relu	3 * 3	384	1	1	(3 * 3 * 256 + 1) * 384=885120	(3 * 3 * 256 + 1) * 384 * 13 * 13=149585280
384 * 13 * 13						
Conv4 + Relu	3 * 3	384	1	1	(3 * 3 * 384 + 1) * 384=1327488	(3 * 3 * 384 + 1) * 384 * 13 * 13=224345472
384 * 13 * 13						
Conv5 + Relu	3 * 3	256	1	1	(3 * 3 * 384 + 1) * 256=884992	(3 * 3 * 384 + 1) * 256 * 13 * 13=149563648
256 * 13 * 13						
Max Pooling	3 * 3		2			
256 * 6 * 6						
Dropout (rate 0.5)						
FC6 + Relu					256 * 6 * 6 * 4096=37748736	256 * 6 * 6 * 4096=37748736
4096						
Dropout (rate 0.5)						
FC7 + Relu					4096 * 4096=16777216	4096 * 4096=16777216
4096						
FC8 + Relu					4096 * 1000=4096000	4096 * 1000=4096000
1000 classes						
Overall					62369152=62.3 million	1135906176=1.1 billion
Conv VS FC					Conv:3.7million (6%) , FC: 58.6 million (94%)	Conv: 1.08 billion (95%) , FC: 58.6 million (5%)

Figura 18: Parametros de la arquitectura AlexNet [37]

3.3. VGG16

VGG16 es una red neuronal convolucional propuesta en 2014 por K. Simonyan and A. Zisserman de la Universidad de Oxford en su estudio "Very Deep Convolutional Networks for Large-Scale Image Recognition". Este modelo consta de alrededor de 138 millones de parámetros y consigue un 92.7 por ciento de precisión sobre ImageNet, un dataset de más de 14 millones de imágenes. VGG fue entrenada durante semanas utilizando la GPU NVIDIA Titan Black.

Se trata de una red neuronal posterior al lanzamiento de AlexNet que supuso un avance porque añadió un aspecto importante sobre el que centrarse, la profundidad de la CNN. VGG16, como su nombre indica, contiene 16 capas donde los pesos y bias son aprendidos. De estos, un total de 13 de estas capas son capas convolucionales enlazadas las unas con las otras y otras 3 capas densas para la clasificación. Esto hace una mejora sobre AlexNet sustituyendo el tamaño de los filtros. Las características son obtenidas por max pooling el cual es aplicado en varias etapas dentro de la arquitectura.

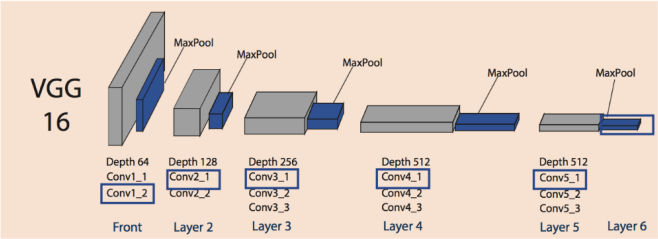


Figura 19: Arquitectura de VGG16

La entrada a la red la imagen tiene dimensiones 224x224x3. Para obtener estas dimensiones, se utilizó un preprocesamiento que consiste en un recorce central con dimensiones 224x224 para mantener el tamaño de la imagen de entrada consistente [38].

Las capas convolucionales tienen un tamaño del filtro de 3x3. Este tamaño es el mínimo tamaño posible que sigue capturando izquierda, derecha, arriba y abajo, una gran diferencia sobre AlexNet. A su vez, existen filtros convolucionales de tamaño 1x1 que actúan como una transformación lineal de la entrada, seguida de una operación ReLU.

Las primeras 2 capas tienen 64 canales de filtros de tamaño 3x3 y mismo padding.

Después de una capa de max pooling con stride (2,2), dos capas convolucionales con 256 filtros de tamaño 3x3. Otra capa con un pooling de stride (2,2) igual que la anterior. Más adelante otros 2 sets formados por 3 capas convolucionales y una de pooling. Cada una de ellas tiene 512 filtros de tamaño 2x2 y el mismo padding. Algunas de las capas también utilizan filtros de tamaño 1x1 hechos para manipular el número de canales de entrada. Después de cada capa convolucional hay un padding de 1 pixxel para prevenir características espaciales en la imagen.

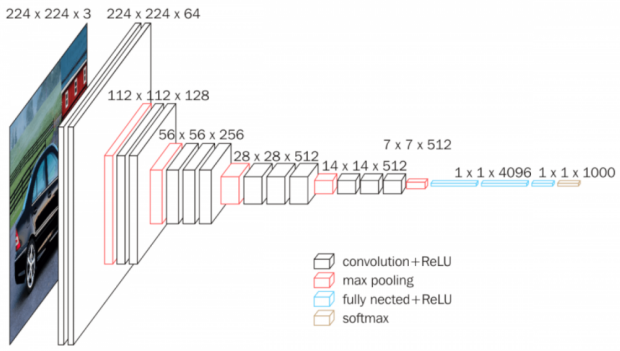


Figura 20: Arquitectura de VGG16

Después de la pila de convoluciones y pooling obtenemos un mapa de características (7, 7, 512). Se convierte esta salida en un vector de características (1, 25088). A continuación hay 3 capas completamente conectadas. La primera capa usa la entrada del ultimo vector y devuelve un vector de (1, 4096). La segunda capa devuelve un vector de las mismas dimensiones pero en la tercera capa se devuelven solamente 1000 canales correspondientes a las distintas clases que componen la red neuronal. Por último, tras pasar por la 3ª capa completamente conectada se procede a una capa de clasificación Softmax en orden de normalizar la clasificación del vector. Todas las capas ocultas usan ReLU como su función de activación. ReLU es computacionalmente más eficiente porque tiene un aprendizaje más rápido.

3.4. ImageNet

ImageNet es un dataset de más de 15 millones de imágenes de alta resolución etiquetadas pertenecientes a aproximadamente 22 mil categorías. Se trata de una base de datos de imágenes organizada según la jerarquía WorldNet [39], en donde cada nodo de la jerarquía es representado por cientos y miles de imágenes con sus correspondientes anotaciones indicando el contenido de las imágenes.

Desde 2010, el proyecto ImageNet lleva a cabo un concurso anual de software, el Desafío de reconocimiento visual a gran escala ImageNet (ILSVRC), donde los programas de software compiten para clasificar y detectar correctamente objetos y escenas.

La gran cantidad de clases que nos proporciona ImageNet relacionadas con los objetivos del estudio hacen que se haya elegido este dataset de cara a entrenar tanto la red AlexNet como VGG16 antes de aplicar la técnica de transfer learning.

3.5. Preprocesamiento

Antes de empezar a operar, se realiza un preprocesamiento que consta de una re-dimensión de las dimensiones de entrada de una imagen a 224x224 píxeles como requieren la mayoría de redes preentrenadas y en este caso AlexNet y VGG16.

A su vez, es útil configurar las transformaciones de datos. Esencialmente, las transformaciones de datos permiten entrenar en muchas variaciones de las imágenes originales que se recortan de diferentes maneras o se rotan de diferentes maneras, además de poder convertir los arrays en tensores, como se ha comentado anteriormente.

Las redes pre-entrenadas fueron entrenadas en el conjunto de datos de ImageNet donde cada canal de color se normalizó por separado. Para ambos conjuntos, se normalizaron los medios y las desviaciones estándar de las imágenes a lo que la red espera. Para las medias, es [0.485, 0.456, 0.406] y para las desviaciones estándar [0.229, 0.224, 0.225], calculadas a partir de las imágenes de ImageNet. Estos valores cambiarán cada canal de color para que se centre en 0 y oscile entre -1 y 1.

El conjunto de validación se utiliza para medir el rendimiento del modelo en datos que aún no se han visto. Para esto, no se desea ninguna transformación de escala o rotación, pero se debe cambiar el tamaño original de la imagen.

3.6. Mi base de datos

Como se ha comentado anteriormente, a la hora de elegir cómo se quiere entrenar la red neuronal, se ha escogido el método que usará una red pre-entrenada con el dataset de imágenes de ImageNet ya que las categorías que esta base de datos incluye resuelven el problema que se está tratando en este estudio y el modelo es entrenado con un gran número de imágenes.

Para la posterior extracción de características se ha utilizado en primera instancia un dataset formado por 2 clases. Una de las clases está formada por imágenes de un domicilio mientras que la otra clase recoge un recorrido a pie de la ciudad de Tres Cantos.

Este dataset está dividido en una parte de train, donde se suman 200 imágenes de ambas clases y una parte de validación que recoge otras 100 imágenes. Por último, existe una parte de test formada por 100 imágenes con la diferencia que estas no están clasificadas según la clase a la que pertenecen.

Las imágenes que pertenecen a la fase de validación son imágenes similares a las utilizadas durante el entrenamiento con la diferencia que estas presentan algura variación. Con las imágenes de un paseo al aire libre se ha utilizado el trayecto de X a Y para tomar las imágenes de entrenamiento y la etapa de validación estaba formada por imágenes haciendo el recorrido Y a X, muy similar pero que presenta diferencias que se aprovecharán para sacar conclusiones tanto de la extracción de nuevos lugares como del reconocimiento de las escenas.

La clase formada por las imágenes correspondientes a las distintas habitaciones y espacios de una casa, ha sido extraida del estudio de estimación de precios de la viviendas en función de características visuales y textuales [40].

Figura 21: Dataset Exterior

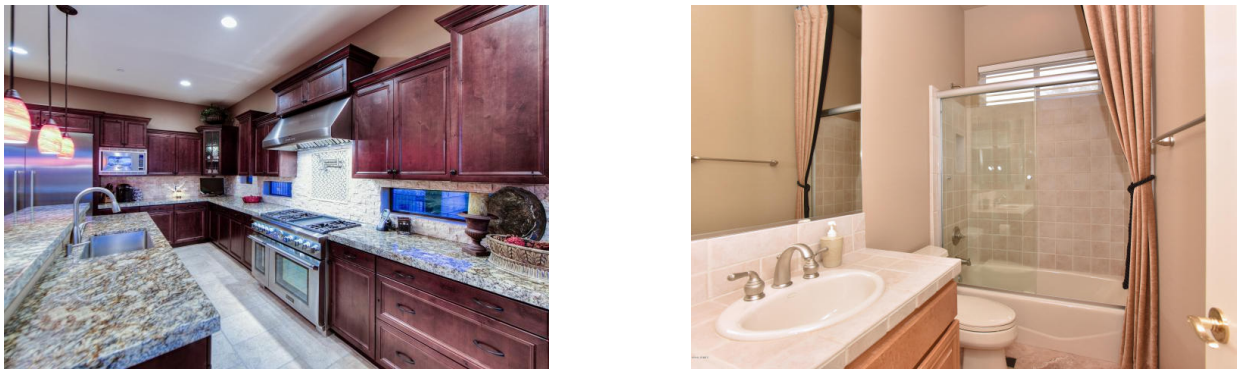


Figura 22: Dataset Interior

3.7. Extracción de características

La extracción de características es un proceso de reducción de dimensionalidad en el que un set inicial de datos sin procesar se reduce a grupos más precisos y manejables para el procesamiento. Una de las mayores características de estos grandes conjuntos de datos es que requieren un gran coste computacional y por ese motivo se ha elegido el uso de tensores en lugar de los clásicos arrays numpy ya que estos nos permiten utilizar la GPU para el cálculo de estas operaciones. La extracción de características es el nombre de los métodos que seleccionan y combinan variables en características, reduciendo la cantidad de datos que deben procesarse, mientras que aún describen de manera precisa y completa el conjunto de datos original.

Este proceso es útil cuando se necesita reducir la cantidad de recursos necesarios para el procesamiento sin perder información importante o relevante. La extracción a su vez puede reducir la cantidad de datos redundantes para un análisis dado. Estas reducciones de datos suponen un ahorro en los esfuerzos de la máquina que se traduce una velocidad de aprendizaje mayor.

Una de las grandes dificultades de cara a la implementación del código ha sido el uso de operaciones vectoriales lo que permite una mayor eficiencia de nuestro algoritmo y menor tiempo de procesamiento. Esto supone una gran ventaja tanto para ahorrar recursos así como una forma más directa de implementar el código.

Ya se ha mencionado que uno de los objetivos de este trabajo es que nuestro modelo sea capaz de reconocer si las imágenes que se están pasando corresponden a imágenes que ya se han visto antes o que por el contrario son nuevas y si se trata de este tipo, añadirlas en una tabla de lugares que vaya aumentando según reconozca características completamente diferentes.

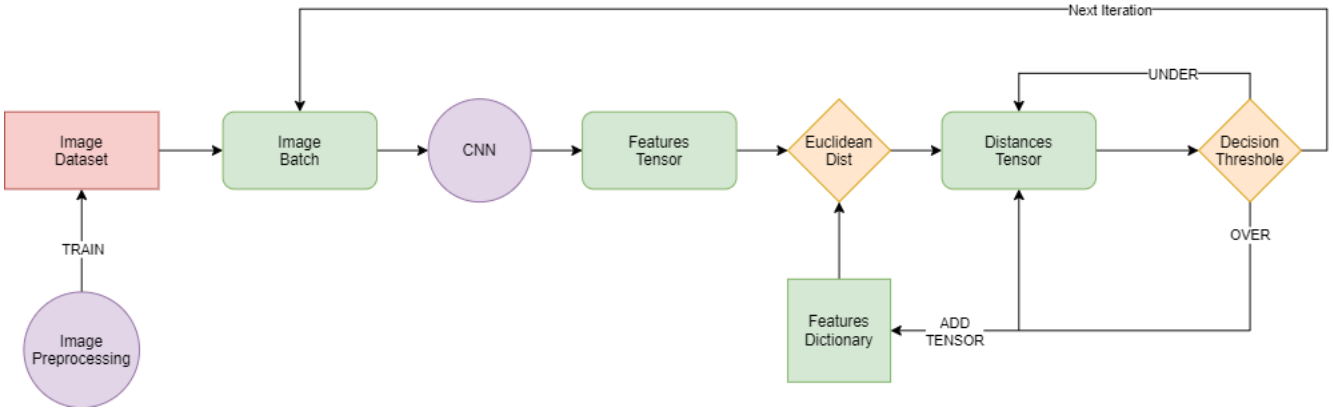


Figura 23: Diagrama de la extracción de características

Como muestra el diagrama anterior, el primer paso es un preprocesamiento de las imágenes que tenemos del dataset, únicamente aplicado en las fases de entrenamiento y validación. Este procesamiento previo de la imagen recoge transformaciones de normalización y centrado de la imagen, vistas las mejoras que se producen al aplicar este procesamiento standar utilizado en este tipo de redes neuronales.

Una vez tenga lugar el procesamiento previo a la imagen, si aplica, se carga el dataset de imágenes almacenadas en un mini-batch que permite que las imágenes sean procesadas de forma paralela. A continuación se importan los datos de entrada a nuestro modelo de la red neuronal. En la salida se obtiene tensor de dimensiones (X,N) donde X indica el número de imágenes que se están pasando en el mini batch mientras que N representa el número de clases que devuelve la capa convolucional correspondiente.

Como se comentará en el próximo apartado, se han utilizado distintas salidas de las capas convolucionales de los modelos de AlexNet y VGG16 para poder hacer una comparativa y sacar conclusiones acerca de dónde es más preciso la función de extracción de características.

El objetivo es crear una tabla de lugares (diccionario) que almacene todas las características de cada imagen siempre y cuando todas estas características superen un umbral de decisión. En primera instancia, este está vacío, por lo que se introducirá el primer tensor que se procese como punto de partida.

Para averiguar si se supera ese umbral de decisión se calculan las distancias euclídeas entre los tensores de la salida de nuestro modelo que esté iterando el mini-batch y el diccionario de lugares. Cada vez que un tensor supere las condiciones necesarias para que se considere como un "nuevo lugar"será añadido a la tabla de lugares. Si un tensor no supera el umbral, se comparará el siguiente tensor de distancias hasta que no haya más y se volverá a calcular las distancias euclídeas en una nueva iteración con distintas imágenes en el mini-batch y el diccionario actualizado con los nuevos lugares incluidos.

El valor del umbral de decisión ha sido obtenido gracias a las diferentes pruebas y experimentos que se han realizado. Como se ha comentado, en nuestro diccionario se pretende que los vectores de características de las imágenes que se envía sean suficientemente diferentes como para que su distancia supere el umbral y de este modo se almacene en el diccionario.

3.8. Algoritmo K-means

K-means es un algoritmo no supervisado de clustering. Esto quiere decir, que al ser no supervisado, no se tendrá información de las etiquetas de los datos de entrenamiento. Los algoritmos de clustering se utilizan para agrupar datos de los que se desconocen sus características. Estos grupos crean puntos centrales y jerarquías para diferenciar grupos y descubrir características comunes por cercanía.

El algoritmo de Clustering K-means es uno de los más usados para encontrar grupos ocultos, o sospechados en teoría sobre un conjunto de datos no etiquetado. Esto puede servir para confirmar -o desterrar- alguna teoría que teníamos asumida de nuestros datos. También puede ayudar a descubrir relaciones entre conjuntos de datos, que de manera manual, no se hubieran reconocido. Una vez que el algoritmo ha ejecutado y obtenido las etiquetas, será fácil clasificar nuevos valores o muestras entre los grupos obtenidos.

El objetivo de k-means es encontrar k grupos de clusters entre los datos. El algoritmo trabaja iterativamente para asignar a cada punto uno de los k grupos basados en sus características, es decir, se agrupan en base a la similitud de sus características.

Los grupos se van definiendo de manera progresiva, lo que quiere decir que se va ajustando la posición de estos centroides en cada iteración del proceso hasta que converge el algoritmo.

El algoritmo utiliza un proceso iterativo en el que se van ajustando los grupos para producir el resultado final. A la entrada del algoritmo k-means, este recibirá nuestra tabla de lugares (diccionario) con cada uno de los vectores considerados como lugares diferentes, de dimensiones iguales a la salida de la capa convolucional que se esté aplicando en cada caso. También se tiene que definir el numero de clases k. Las posiciones iniciales de los centroides serán asignadas de manera aleatoria y conincidirán en dimensiones con los tensores de la tabla de lugares. PAra este estudio, se ha utilizado un valor de k igual a 2. Es decir, sólo se están evaluando las clases interior y exterior como se utiliza en el artículo de referencia visto para la realización de este trabajo.

En una primera parte, el algoritmo calcula la distancia euclídea de los k centroides a todos los vectores de nuestra tabla de lugares. Una vez hecho, se ordena de forma que se determine que centroide es el que está más cerca (o su distancia es menor) de cada uno de los vectores de la tabla de lugares. Cuando se hayan reordenado las distancias, se procede a recalcular los puntos originales de los centroides utilizando el promedio de los vectores del diccionario que estén más cerca de este centroide. Esto se realizará hasta que los centroides dejen de variar y mantengan inmóviles sus coordenadas.

Este promedio se calcula teniendo en cuenta sólo los tensores que estén más cerca del centroide correspondiente. Es decir, se considerará que el tensor en análisis tiene que estar más cerca de un centroide que del otro y por tanto sólo se utilizará para recalcular el centroide del que estaba más cerca. Una vez se hayan centrado todos los tensores correspondientes a las k clases propuestas, se realiza una operación para comprobar la igualdad de lso tensores centroides actuales y los de la siguiente iteración.

Se está tomando una precisión de 10e-6 para estimar si los tensores considerados como centroides se consideran iguales.

3.9. Reconocimiento Imágenes Nuevas

Una vez se hayan implementado las partes del reconocimiento de lugares con la extracción de las caracteristicas y se haya generado el algoritmo de clasificación k-means se procede a utilizar un nuevo dataset de imágenes, en este caso se está aplicando la parte de validación del dataset propio. Esta parte del dataset está formada por 50 imágenes de ambientes exteriores y otras 50 de ambientes interiores (dentro de un domicilio).

El objetivo de esta parte del proyecto es ver ante nuevas imágenes cómo se comporta nuestro diccionario de lugares, si está reconociendo correctamente el lugar más cercano de la nueva imagen a la tabla de lugares y si esa imagen bajo análisis es clasificada correctamente bajo el centroide que muestre una menor distancia.

En primera instancia, se sigue la estructura del extractor de características formada por un dataloader encargado de pasar en un mini-batch un conjunto de imágenes. Se ha utilizado un batch de tamaño 10. Una de las razones principales de esta decisión ha sido el buscar simplificar al máximo los procesos iterativos y evitar errores en las iteraciones que no estén completas.

Una vez se hayan enviado correctamente los datos de las imágenes, estas pasan por el modelo de nuestra red neuronal convolucional y sobre la salida se calcula la distancia euclídea con todos los puntos del diccionario de lugares. Una vez se hayan calculado estas distancias, se procede a ordenarlas y se utiliza solamente la distancia más pequeña de cada imagen del mini-batch a cada uno de los puntos del diccionario.

Esto se realiza de manera iterativa sobre todas las imágenes hasta obtener a la salida de esta función un tensor que tenga el índice del tensor de la tabla de lugares más cercano a la imagen que se ha tratado. Además, se ha calculado de una manera similar el centroide que está más cerca de la imagen.

Con estos datos, se tiene suficiente información como para comprobar la funcionalidad de los algoritmos sobre las distintas capas convolucionales de los modelos pre-entrenados, así como un estudio de todas las variables que pueden configurarse y su relación tanto directa como indirecta a los resultados obtenidos que se plantea a continuación.

4. Pruebas y resultados

4.1. Introducción

En este capítulo se estudiarán los distintos resultados obtenidos tras la implementación del reconocedor y el clasificador comentados previamente.

La estructura de esta sección estará dividida en dos bloques diferentes, uno asociado a cada una de las redes convolucionales que se han utilizado (AlexNet y VGG16). En cada uno de estos bloques se analizarán las distintas capas que se han utilizado como salida del modelo de CNN, se comprobarán cuáles son las capas más eficientes para el caso práctico que se está estudiando así como el por qué de los valores que se han utilizado para cada una de las variables que se han utilizado para marcar los umbrales usados.

Como se acaba de comentar, se ha utilizado un umbral para verificar la distancia entre los tensores de las imágenes del mini-batch y los apuntados en tabla de lugares. El valor de este umbral ha sido tomado a base de prueba y error y se le ha dado un valor numérico que ha ido cambiando a lo largo de las distintas pruebas efectuadas para obtener en todas ellas un tamaño de diccionario similar.

Debido a las pruebas realizadas se ha visto que si se utiliza un valor muy elevado para así solo añadir nuevos lugares a nuestro diccionario que sean completamente diferentes a los que se tienen guardados, el reconocedor no será capaz de aprender nuevos lugares. Esto sucede ya que cuanto más se aumente el valor del umbral, será más complicado que se añada un nuevo tensor asociado a una imagen. Si por ejemplo la distancia euclídea entre muchas características del tensor de la tabla de lugares y la imagen tratada es elevada, lo que significa que esas características no están correladas, podría no añadirse esta imagen ya que con que una característica no supere el umbral, ese tensor no será añadido.

De la misma forma, no podemos tener un valor del umbral muy pequeño ya que esto provocaría que con que hubiera una mínima diferencia, el reconocedor la trataría como un nuevo lugar cuando simplemente podría tratarse de una imagen ligeramente distinta de la otra aunque ambas hubieran sido sacadas de un dataset formado por fotogramas de un mismo recorrido.

A continuación se describirán los resultados obtenidos en cada una de las distintas capas que se han utilizado. Hay que tener en cuenta que cada salida de nuestro modelo puede tener distinto número de características tanto de salida como de entrada. Más abajo se muestra una imagen de las dimensiones de cada una de las salidas de estas capas.

4.2. Arquitectura VGG16

La arquitectura de VGG16 sigue la siguiente estructura. En ella se puede apreciar una primera fase convolucional seguida de un clasificador para redimensionar las salidas obtenidas de la extracción de características. A continuación se ha explicado de manera breve las distintas etapas que forman esta arquitectura y se ha expuesto un caso para cada conjunto de capas.

```
summary(vgg, (3, 224, 224))

----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 224, 224]           1,792
              ReLU-2         [-1, 64, 224, 224]               0
            Conv2d-3         [-1, 64, 224, 224]          36,928
              ReLU-4         [-1, 64, 224, 224]               0
         MaxPool2d-5         [-1, 64, 112, 112]               0
            Conv2d-6        [-1, 128, 112, 112]          73,856
              ReLU-7        [-1, 128, 112, 112]               0
            Conv2d-8        [-1, 128, 112, 112]         147,584
              ReLU-9        [-1, 128, 112, 112]               0
        MaxPool2d-10          [-1, 128, 56, 56]               0
           Conv2d-11          [-1, 256, 56, 56]         295,168
             ReLU-12          [-1, 256, 56, 56]               0
           Conv2d-13          [-1, 256, 56, 56]         590,080
             ReLU-14          [-1, 256, 56, 56]               0
           Conv2d-15          [-1, 256, 56, 56]         590,080
             ReLU-16          [-1, 256, 56, 56]               0
        MaxPool2d-17          [-1, 256, 28, 28]               0
           Conv2d-18          [-1, 512, 28, 28]       1,180,160
             ReLU-19          [-1, 512, 28, 28]               0
           Conv2d-20          [-1, 512, 28, 28]       2,359,808
             ReLU-21          [-1, 512, 28, 28]               0
           Conv2d-22          [-1, 512, 28, 28]       2,359,808
             ReLU-23          [-1, 512, 28, 28]               0
        MaxPool2d-24          [-1, 512, 14, 14]               0
           Conv2d-25          [-1, 512, 14, 14]       2,359,808
             ReLU-26          [-1, 512, 14, 14]               0
           Conv2d-27          [-1, 512, 14, 14]       2,359,808
             ReLU-28          [-1, 512, 14, 14]               0
           Conv2d-29          [-1, 512, 14, 14]       2,359,808
             ReLU-30          [-1, 512, 14, 14]               0
        MaxPool2d-31            [-1, 512, 7, 7]               0
           Linear-32                 [-1, 4096]     102,764,544
             ReLU-33                 [-1, 4096]               0
          Dropout-34                 [-1, 4096]               0
           Linear-35                 [-1, 4096]      16,781,312
             ReLU-36                 [-1, 4096]               0
          Dropout-37                 [-1, 4096]               0
           Linear-38                 [-1, 1000]       4,097,000
================================================================
```

Figura 24: Características, parametros y dimensiones de VGG16

El comportamiento varía en gran medida dependiendo de la profundidad de cada capa. Las capas iniciales (bloques 1 y 2) mantienen la mayoría de las características de la imagen. Los filtros convolucionales son activados en cada parte de la imagen de entrada. Esto da una idea que estos filtros iniciales pueden ser detectores de límites primitivos (se puede construir una imagen compleja a partir de pequeños límites y orientaciones puestos juntos).

Según se avanza (bloques 3 y 4) las características extraidas por estos filtros se vuelven visualmente poco interpretables. Una de las razones por las que se debe esto es que se está extrayendo información de la imagen original y tratar de convertirlo a un dominio más cercano a la clasificación de la salida del modelo.

En el bloque 5 (especialmente en "block-conv3") se pueden apreciar muchas salidas de los bloques convolucionales que están vacíos. Esto puede significar que los patrones codificados por los filtros no han sido encontrados en la imagen de entrada. Probablemente, estos patrones se deben a formas complejas no representados en la imagen.

4.2.1. Salida completamente conectada Lineal 38

Se ha empezado por medir las características a la salida de la última capa completamente conectada del clasificador. En este caso se tendrán tensores de 1000 parámetros. Al utilizar el umbral marcado anteriormente con un valor numérico de 50, se ha creado un diccionario de 119 lugares distintos a partir de las 200 imágenes de entrenamiento sin etiquetarlas de ninguna manera. Posteriormente, se ha clasificado a partir del algoritmo de clustering observando que se ha obtenido para el conjunto de validación una distribución aproximadamente del 50 por ciento, lo que supone una buena estimación ya que este set de imágenes de validación estaba formado por 50 imágenes pertenecientes a cada una de las dos clases.

Si se continúa con el análisis, es posible observar que estos resultados no son tan 'ideales' como se buscan. Al no hacer un ordenamiento previo de la entrada de las imágenes, sino que se envían siguiendo un orden lógico, permite ver de manera más visual la situación real en cuanto a la clasificación. Se puede observar cómo existen algunos falsos positivos reconociendo una estancia de la casa como perteneciente a la clase de imágenes del exterior. Esto se ha podido comprobar ya que dentro del dataset de imágenes interiores se han encontrado algunas fotografías que muestran el jardín/terraza de una casa y por ello se ha identificado como perteneciente a la otra clase.

A continuación se ha hecho una comparativa con los resultados que más se han repetido dentro de los distintos tensores del diccionario de lugares. Se puede ver cómo para las imágenes del interior de una vivienda es capaz de reconocer en algunos casos, menos en uno, distintas imágenes de cuartos de baño, lo que significa que si que está aprendiendo y que se están diferenciando las distintas estancias de una vivienda aunque posiblemente en esta última capa no se esté siendo lo suficientemente preciso.

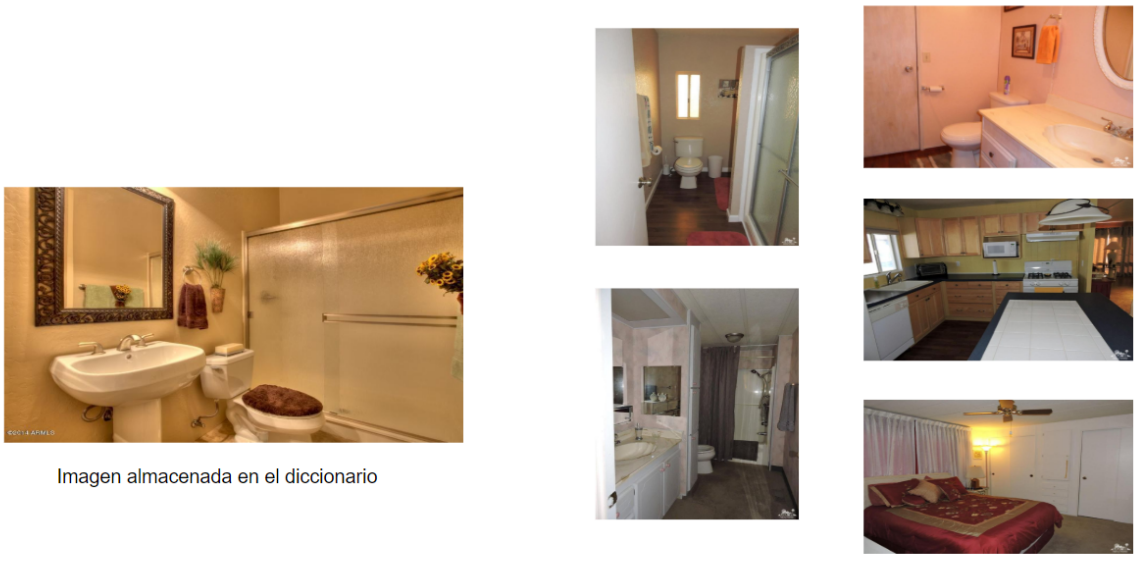


Figura 25: Escenas interiores de la etapa final del clasificador

Con respecto a las imágenes del dataset de exteriores, se mostrará de la misma manera la imagen más repetida según los sitios almacenados en el diccionario de lugares. En este caso se puede observar cómo el algoritmo de reconocimiento de lugar no está funcionando como debería ya que se

están reconociendo solamente estructuras. Como se aprecia, se están reconociendo columnas, que se interpretan en el conjunto de imágenes validación como árboles.

Figura 26: Escenas exteriores de la etapa final del clasificador

4.2.2. Capa completamente conectada Linear35

Este segundo análisis se han aumentado las dimensiones de los parametros al no utilizar la última capa de redimensión del clasificador que reduce los parámetros de 4096 a 1000. Con este cambio, se aumentan las posibilidades de encontrar nuevas características asociadas a las imágenes evaluadas.

Si uno se fija en los resultados obtenidos del clasificador K-means, se puede observar cómo todas las imágenes del entorno de validación correspondientes a la clase interior han sido clasificadas correctamente. Con el otro conjunto de imágenes, se han tenido más dificultades para clasificarlas correctamente, obteniendo unos resultados mucho peores que en las pruebas hechas a la salida del modelo después de la reducción de las dimensiones.

Esto mismo se puede apreciar más claramente en el reconocedor de lugares. En él, se puede ver cómo las características extraidas en este nivel son mucho más precisas para un dataset que para el otro. Al observar las imágenes que es capaz de asociar tanto de una como de otra clase, es fácil darse cuenta que la precisión con la que se reconocen estancias de un domicilio es realmente notable, siendo capaz de distinguir en la mayoría lugares del diccionario a donde pertenece. Los lugares del diccionario del dataset de interiores en su gran mayoría enlazan correctamente imágenes de un mismo tipo de habitaciones. Se puede apreciar en la siguiente imagen una agrupación de uno de los lugares del diccionario.

Figura 27: Escenas interiores de la etapa intermedia de clasificador

Por otro lado, es igualmente fácil apreciar que el reconocedor de clases exteriores no está funcionando como debería. En el ejemplo que se ha adjuntado de esta clase, se ha seleccionando el lugar más repetido en todo el conjunto de validación. En esta última clase se ve cómo no está detectando correctamente ciertos parámetros y está asociando lugares que no tienen correlación entre ellos.

Figura 28: Escenas exteriores de la etapa intermedia de clasificador

4.2.3. Capa convolucional 5-3

Una vez se ha eliminado la parte del clasificador de VGG16, se procederá a realizar el mismo análisis aunque sobre las capas convolucionales secuenciales. Una de las principales características cuando se contempla la arquitectura de la red sin utilizar ningún clasificador, es cómo la cantidad de parámetros se multiplica exponencialmente hasta el punto de volverse una matriz de características muy poco intuitiva.

Uno de los principales problemas que desencadena el tener un número tan grande de variables es que a la hora de construir el diccionario de lugares, si se aplica el mismo umbral que se ha utilizado en las anteriores ocasiones, provoca que no sea capaz de distinguir las imágenes de entrada y las identifique todas como si se tratasen de lugares nuevos a añadir en la tabla.

Se ha contemplado cómo este umbral se trata de un parámetro de vital importancia ya que gracias a él se podrá ajustar de forma eficiente para el caso de estudio que se quiera y se calibrará en función del número de parametros que se necesiten. En este caso, se ha optado por tener un tamaño de diccionario de lugares similar a los utilizados en pruebas anteriores. Para ello, se ha ido variando el valor que recibe la variable del umbral hasta obtener un valor numérico de 600.

Como se puede observar, se ha multiplicado este valor por aproximadamente 10 veces lo que solía ser en las primeras pruebas. Esto en primera instancia puede parecer excesivo el factor de multiplicación, pero si se tiene en cuenta que se ha pasado de tener 1000/4096 parámetros a 100352 se puede comprender el aumento del umbral.

En cuanto a los resultados obtenidos en esta capa de la red neuronal el clasificador es capaz de distinguir correctamente entre las 2 clases seleccionadas aunque con una efectividad mucho menor que en los casos anteriores. Es posible observar cómo las asociaciones erróneas se presentan en ambos datasets de imágenes, tanto en el conjunto de interiores como en el exterior.

A la hora de comparar los resultados obtenidos en relación con el diccionario de lugares se ve claramente una tendencia en la que se está identificando erróneamente las imágenes de validación con las almacenadas previamente como lugares nuevos.

Observando cómo las imágenes del dataset de interiores son asociadas, se puede afirmar que en la mayoría de casos se están asociando erróneamente a la hora de clasificar con las imágenes del conjunto de exteriores del diccionario de lugares. No solo ello, sino que tampoco se consigue la precisión encontrada en capas anteriores para asociar e interpretar las distintas estancias de un domicilio.

Figura 29: Escenas interiores en la capa conv. 5-3

Para el otro conjunto de imágenes, los resultados muestran cómo se está asociando la mayoría de imágenes a un lugar que no corresponde con la información visual que contiene. No solamente eso, sino que además asocia imágenes del dataset de interiores. Una de las posibilidades de que esto suceda es que en este nivel de la red la mayoría de características no contienen ninguna información útil y al tener que elegir siempre la que tenga una mayor similitud escoge aquellas que al menos tienen algo de información aunque la distancia que se ha calculado sea muy grande.

Figura 30: Escenas exteriores en la capa conv. 5-3

4.2.4. capa convolucional 4-3

En esta capa, el número de parámetros extraidos aumenta exponencialmente hasta situarse en 100352. Para obtener un tamaño del diccionario de lugares que ronde los 120 casos, se ha propuesto un umbral con valor de 800. En cuanto a los resultados obtenidos gracias al clasificador K-Means se observa cómo todas las imágenes del conjunto de exteriores han sido clasificadas correctamente pero con el otro conjunto se han obtenido una cantidad notable de imágenes mal clasificadas. Esto nos da una idea de la precisión de esta capa.

Figura 31: Escenas exteriores en la capa conv. 4-3

En esta capa, se obtienen resultados muy pobres, donde un lugar del diccionario, ha sido identificado en más de 75 imágenes de la etapa de validación. Uno de los problemas más preocupantes de esta capa es que no se ha asociado ninguna imagen del conjunto de interiores de validación con las almacenadas dentro del diccionario de lugares obtenido con el extractor de características. Esto deja patente que no se debe utilizar esta capa para realizar este análisis ya que los resultados como se comprueba en las imágenes ilustradas no muestran ninguna característica común.

Figura 32: Escenas exteriores en la capa conv. 4-3

4.2.5. capa convolucional 3-3

En la capa convolucional 3-3 se está utilizando un umbral de decisión con valor 7000 para obtener un tamaño de diccionario de 124. Los resultados obtenidos en la clasificación han sido mejores que los obtenidos en el resto de capas, con un porcentaje muy bajo de imágenes mal clasificadas. Las imágenes asociadas del conjunto de exteriores muestran buena respuesta aunque se incluyen algunas imágenes con escenas de domicilios asociadas a lugares exteriores. A pesar de ello, muchas de las imágenes muestran elementos comunes característicos que han permitido tal relación de las imágenes.

Para el conjunto de interiores, solo se ha obtenido un lugar que se ha reconocido en multitud de ocasiones aunque todos los resultados pertenecen a conjuntos de imágenes interiores salvo un par de casos mal identificados. En conclusión, se podría decir que se han obtenido mejores resultados para el conjunto de exteriores que para el de interiores.

Figura 33: Escenas exteriores en la capa conv. 3-3

Figura 34: Escenas exteriores en la capa conv. 3-3

4.2.6. capa convolucional 2-2

En esta capa del modelo de la red neuronal convolucional se han obtenido unos resultados muy por debajo de los resultados de otras capas en la clasificación de imágenes pertenecientes al conjunto de interiores. Para el dataset de entornos exteriores se han obtenido todas las imágenes bien clasificadas.

El reconocimiento de imágenes del diccionario de lugares no ha funcionado correctamente. Es más, se han encontrado simplemente 2 lugares en los que se ha dividido todas las imágenes de la etapa de validación. Esto es un problema, ya que aunque si se han asociado todas las imágenes de cada dataset estas lo han hecho erróneamente, ya que por la experiencia de pruebas anteriores hay imágenes más relacionadas con las de la etapa de validación que las mostradas. Además, se han entremezclado a su vez imágenes de ambos conjuntos en los dos lugares reconocidos.

Figura 35: Escenas exteriores en la capa conv. 2-2

Figura 36: Escenas exteriores en la capa conv. 2-2

4.2.7. Capa Convolucional 1-2

Para analizar los parámetros obtenidos a la salida de esta capa convolucional, se puede observar cómo se tiene un error debido a la cantidad de parámetros que se están extrayendo. Para poder analizar los resultados obtenidos en esta capa, se ha redimensionado a la mitad el tamaño de las filas y las columnas aplicando una etapa de MaxPool2d. Realizando esta operación una vez, el número de parametros obtenido a la salida es soportado por CUDA y se puede realizar el análisis como el los casos previos.

A pesar de ello, esto no ha resultado una idea demasiado buena, ya que tanto el clasificador según K-Means cómo el reconocedor de lugares han asignado todas las imágenes como si se tratasen de escenas exteriores y se han asociado todas las imágenes de la etapa de validación a un mismo lugar guardado en el diccionario de imágenes. Por esta razón, no se han adjuntado ninguna imagen ya que no esta funcionando correctamente nuestro algoritmo en la redimensión efectuada en esta capa y por tanto no se pueden hacer comparaciones con los resultados.

4.3. Arquitectura AlexNet

4.3.1. Capa completamente conectada Linear 19

A la salida del clasificador de AlexNet, se extraen 1000 características por cada imagen que se envíe y se almacenan en un tensor. Para esta arquitectura se ha utilizado un umbral de decisión de inclusión con un valor de 60 para añadir al diccionario de imágenes lo que ha resultado en 121 lugares distintos reconocidos por el extractor de características.

A la hora de la clasificación según los clusters, se han obtenido unos resultados de 61 imágenes clasificadas como escenas exteriores y las 39 restantes como imágenes interiores. Categorizar que se han cometido errores en la clasificación es asumible debido al modelo que se ha utilizado con los dataset y se ha comentado en la salida del clasificador VGG16.

A la hora de analizar las imágenes asociadas de la etapa de validación a las pertenecientes al diccionario de lugares se ha encontrado algunos resultados interesantes. Para empezar, los lugares almacenados que representan imágenes de un domicilio guardan mucha similitud y se puede comprobar cómo aquellos que más se repiten son capaces de detectar una misma escena de una habitación de una vivienda.

Figura 37: Escenas interiores a la salida de la capa completamente conectada

Para las imágenes del exterior los resultados han sido muy diferentes a los obtenidos en esta profundidad de la red VGG16. Queda patente cómo es menor el número de imágenes asociadas, aunque las que enlaza se repiten de forma mucho más habitual que en otras pruebas realizadas. Otro dato significativo a tener en cuenta es que en estos casos la imagen asociada se repite sucesivamente sobre ciertas imágenes, con lo que se puede afirmar que está clasificando correctamente 2 escenas similares (tomadas las fotografías con un par de metros de separación). Esto es algo muy positivo ya que aunque pueda no ser exactamente igual la imagen usada para crear el diccionario y las usadas en la etapa de validación, está aprendiendo y asociando un lugar del diccionario a varias imágenes contiguas del dataset.

El ejemplo mostrado a continuación recoge el lugar más repetido a lo largo de toda esta prueba que se repite un total de 14 veces. Se han adjuntado algunos de los resultados en donde se puede apreciar lo explicado anteriormente de conjuntos de imágenes contiguas.

Figura 38: Escenas exteriores a la salida de la capa completamente conectada

4.3.2. Capa completamente conectada Linear 17

En esta segunda prueba, se han obtenido de la capa anterior un total de 4096 parametros pertenecientes a características de una imagen. En este caso, se ha tenido que aumentar el valor del umbral ya que estaba clasificando todas las imágenes recibidas en la primera etapa y se ha utilizado un valor numérico 200 para obtener un diccionario de tamaño 126.

Si se observan los resultados de la clasificación de K-means es distinguible cómo se están clasificando un total de 67 imágenes en el cluster que en teoría debe identificar imágenes del exterior o del interior de un domicilio. En los resultados se aprecia que las clasificaciones erróneas se han realizado en el conjunto de fotografías del escenario de interiores. Es posible ver una ligero empeoramiento de la calidad de la clasificación.

Con respecto a la parte de la asociación de imágenes con el diccionario se ha podido ver que a pesar de obtener resultados peores a los recibidos a la salida del clasificador AlexNet, siguen siendo de utilidad además de que incluyen información útil a la hora de guardar características de la imagen.

Figura 39: Escenas exteriores a la salida de la capa completamente conectada

Se empieza a ver cómo las asociaciones no coinciden al mismo nivel que en la capa anterior, aunque todavía con un grado de acierto notable. En los entornos interiores se puede percibir cómo no se repiten tanto los lugares y se cometen fallos en el reconocimiento. Un comportamiento ligeramente diferente tiene el entorno de imágenes exteriores ya que todavía conserva muchos de los principios del clasificador sin quitar ninguna capa y presenta unos resultados más que aceptables en cuanto a la asociación de escenas se refiere.

Figura 40: Escenas interiores a la salida de la capa completamente conectada

4.3.3. Capa convolucional 5

Para esta prueba se eliminará la etapa del clasificador del modelo de la red AlexNet, utilizando solamente la etapa secuencial. Para esta prueba se ha situado el umbral de decisión del extractor de características en 600 para obtener un tamaño de diccionario que sea cercano a 120. En este caso, el número de escenas extraidas por el diccionario es de 117.

Al extraer las características en esta capa, el número de parámetros aumenta de manera exponencial, hasta el punto de utilizar 43264 para esta capa. Esto puede llegar a suponer un problema ya que se está subdividiendo excesivamente las características que posee una imagen, siendo muchos de estos parámetros inútiles.

Los resultados obtenidos a la hora de clasificar con el algoritmo de K-Means muestran cómo no se están dividiendo correctamente y los centroides no han debido de situarse correctamente. A esta afirmación se puede llegar gracias a los resultados obtenidos donde se aprecia un porcentaje alto de imágenes que han sido clasificadas incorrectamente. Esto puede deberse a que erróneamente se vayan actualizando los centroides buscando aquellos que su distancia euclídea sea menor, pero al haber tantos parámetros esto puede hacer confundir al clasificador y realizar una mala organización de los centros.

A la hora de analizar los resultados obtenidos del extractor de características no mejoran a los resultados del clasificador ya que en este caso está entrelazando equivocadamente imágenes de uno y otro dataset. Se observan peores resultados para los lugares almacenados pertenecientes a escenas interiores ya que en el dataset del domicilio al menos es capaz de reconocer que sigue teniendo ciertas características en común para determinar que pertenece al mismo lugar de la imagen del diccionario.

Figura 41: Escenas interiores en la capa convolucional 5

En cuanto a los lugares almacenados en el diccionario y posteriormente evaluados del dataset de imágenes exteriores se puede decir que los resultados son bastante malos, ya que no está cumpliendo ningún patrón al realizar estas asignaciones. En los lugares más repetidos según nuestro clasificador se recogen tanto imágenes de los interiores de una vivienda, como se puede ver en la imagen, así como de lugares exteriores sin relación aparente.

Figura 42: Escenas interiores en la capa convolucional 5

4.3.4. Capa convolucional 4

Para esta prueba se ha utilizado un valor del umbral de decisión 900 para obtener el tamaño del diccionario que se ha utilizado para todas las pruebas. Los resultados obtenidos a la salida de esta capa convolucional tienen unos resultados similares a los anteriores. En cuanto se refiere a la clasificación según las clases es posible dictaminar que el conjunto de imágenes del exterior se han clasificado correctamente, aunque para el dataset de interiores no se ha conseguido clasificar correctamente alrededor de la mitad del mismo.

La asociación de las características no es tan buena, donde para casi todo el conjunto de exteriores se ha asociado la misma imagen guardada en el diccionario de lugares mientras que para las imágenes de interiores apenas existen grandes conjuntos de lugares y se vuelven a mezclar erróneamente en la asignación ambos datasets.

Figura 43: Escenas Exteriores en la capa convolucional 4

Figura 44: Escenas interiores en la capa convolucional 4

4.3.5. Capa convolucional 3

Para la salida de la 2ª capa convolucional se ha utilizado un valor del umbral de decisión 1800 para obtener un tamaño de diccionario 127.

En el cluster según K-means se han detectado alrededor del 75 por ciento de imágenes clasificadas como imágenes del exterior. Se puede observar cómo la gran mayoría de casos erróneos se han dado en el dataset de interiores ya que para el otro dataset, se clasifican correctamente.

En cuanto al reconocimiento de lugares se observa un dato diferencial con respecto al resto de pruebas ya que ha añadido una de las imágenes de entrenamiento pertenecientes al datasets de imágenes de domicilios y se ha repetido en numerosas ocasiones a lo largo de la etapa de validación. Los resultados que se aprecian no son de gran calidad ni similitud, a pesar de ello se observa cómo el estudio cumple con el requisito de reconocimiento de imágenes sin utilizar ningún tipo de etiquetado como es el caso ilustrado a continuación donde ha reconocido un entorno exterior dentro del otro dataset y posteriormente se han asignado imágenes de la etapa de validación con elementos comunes.

Figura 45: Escenas Exteriores en la capa convolucional 3

Los resultados obtenidos para el dataset de interiores es malo comparado con el grado de acierto de otras capas. Muchas de las imágenes de validación se han asignado dentro del diccionario a imágenes de conjuntos exteriores. A pesar de ello, los pocos conjuntos asignados contenían información similar como se puede comprobar en el ejemplo adjunto.

Figura 46: Escenas interiores en la capa convolucional 3

4.3.6. Capa convolucional 2

Figura 47: Escenas Exteriores en la capa convolucional 2

Figura 48: Escenas interiores en la capa convolucional 2

Los resultados obtenidos en el reconocedor de imágenes muestran buenos resultados para el dataset de exteriores como se puede observar a continuación uno de los ejemplos que mejor ha funcionado. Lamentablemente, esto no se ha repetido para el resto de lugares guardados en el diccionario, ya que sobre todo en las imágenes de lugares interiores se han obtenido resultados muy dispares que apenas guardan relación. Para este caso se ha utilizado un umbral de tamaño 2100.

Se puede concluir afirmando que en esta capa se obtienen resultados significativamente superiores para un dataset que para el otro. En futuros estudios se puede estudiar una relación más directa entre las imágenes utilizadas preservando el reconocedor de imágenes.

4.3.7. Capa convolucional 1

En esta capa se ha utilizado un valor umbral de 600. Los resultados obtenidos en el cluster han sido bastante negativos, donde apenas se han clasificado imágenes como pertenecientes al dataset de interiores. Se han obtenido multitud de errores en esta clasificación posiblemente por el elevado número de parámetros que se han extraido en esta capa de la red. Esto ha propiciado que no se hayan actualizado los centroides correctamente clasificando la mayoría de imágenes como más cercana a uno de los dos centros.

Se puede apreciar en la ilustración siguiente la poca relación de la asignación de lugares del diccionario. Para los lugares interiores de nuestro diccionario no ha sido capaz de detectar correctamente casi ningún lugar para este dataset. El ejemplo muestra el único resultado en el que se han mantenido algunas imágenes de ese conjunto, aunque a su vez han detectado imágenes pertenecientes al otro dataset.

Figura 49: Escenas interiores en la capa convolucional 1

Los resultados del otro dataset se han clasificado en pocos lugares almacenados en el diccionario obteniendo muchas imágenes de la etapa de validación agrupadas en estos pocos lugares. Uno de los ejemplos que se aprecian a continuación, el más repetido a lo largo de esta etapa donde se aprecian muchas similitudes entre las ambos conjuntos de imágenes.

Figura 50: Escenas exteriores en la capa convolucional 1

4.4. Conclusiones de los resultados

Tras analizar los distintos comportamientos que produce el algoritmo de extracción de características y el clasificador K-means es posible llegar a la conclusión que el funcionamiento depende en gran medida del dataset que se esté utilizando. Se ha comprobado cómo en distintas capas se obtienen resultados significativamente superiores en un conjunto de imágenes que en otro.

Uno de los objetivos principales de este estudio, basado en el artículo de referencia era, comprender en que profundidad de las capas se generan las características apropiadas para resolver un problema de clasificación y reconocimiento de imágenes. A diferencia de ese estudio, los resultados obtenidos en las capas completamente conectadas, son muy superiores a los obtenidos en las capas convolucionales.

En cuanto al dataset de interiores se refiere, el grado de acierto tanto en AlexNet como en VGG16 no se ha llegado a ver igualado en ninguna de las otras capas.

Si se trata el otro dataset, el de imágenes exteriores, es posible ver cómo los resultados no son tan sencillos de identificar. Aunque a su vez, los mejores resultados han sido en el clasificador, algunas de las capas han presentado mejores observaciones. Esto se puede deber al menor grado de detalle de las imágenes de este dataset. A diferencia, al ser entornos exteriores de campo abierto, las interpretaciones son mucho más ambiguas. Esto es algo común por el simple hecho que es más sencillo reconocer un baño, que esta formado por una ducha, un labavo, un retrete, que un recorrido por la vía pública donde las estructuras y elementos que lo forman van cambiando.

Otro punto a tener en cuenta es la elección de la red utilizada. Los costes computacionales producidos por la red VGG16, más extensa que AlexNet, apenas se ven reflejados en los rendimientos de una y otra. VGG16 supone un mayor uso de los recursos de la GPU, algo remediable con el uso de otra red neuronal pre-entrenada que no tenga tanta profundidad.

En las imágenes mostradas a continuación se presentan los resultados del clasificador donde se ha medido el rendimiento estándar. En concreto, se ha analizado la precisión y exhaustividad/sensibilidad (llamado recall en Inglés).

Para calcular la precisión P del clasificador para una clase C determinada se considera el número de imágenes de C clasificadas correctamente como pertenecientes a C (Verdaderos Positivos) y el de imágenes de otras clases distintas a C clasificadas incorrectamente como pertenecientes a C (Falsos Positivos). La precisión del clasificador para esa clase C es

$$(P = VP/(VP + FP))$$

Para calcular el recallR del clasificador para una clase C determinada se considera el número de imágenes de C clasificadas correctamente como pertenecientes a C (Verdaderos Positivos) y el de imágenes de la clase C clasificadas incorrectamente como pertenecientes a otras clases (Falsos Negativos). El recall"del clasificador para esa clase C es

$$R = VP/(VP + FN)$$

Una vez calculadas estas medidas, se calcula la llamada ”medida F” calculada como

$$F = 2 \cdot P \cdot R/(P + R)$$

Este valor cuanto mayor es, mejor clasifica las imágenes introducidas.

Dataset Interior			Dataset Exterior			AlexNet
Precisión	Recall	Medida F	Precisión	Recall	Medida F	
1	0.78	0.876	0.819	1	0.901	Fully Connected. (Linear 19)
0.971	0.68	0.799	0.753	1	0.859	Fully Connected. (Linear 17)
1	0.54	0.647	0.65	1	0.78	Fully Connected. (Linear 14)
1	0.32	0.48	0,6756	1	0.806	conv. (Conv 5)
0.9545	0.42	0.566	0.6410	1	0.768	conv. (Conv 4)
0.958	0.46	0.621	0.6578	0.54	0.593	conv. (Conv 3)
1	0.22	0.361	0.562	1	0.719	conv. (Conv 2)
0.857	0.12	0.2105	0.526	1	0.689	conv. (Conv 1)

Figura 51: Resultados clasificador K-means para AlexNet

Dataset Interior			Dataset Exterior			VGG16
Precisión	Recall	Medida F	Precisión	Recall	Medida F	
1	0.88	0.936	0.893	1	0.945	Fully Connected (Linear 38)
1	0.78	0.876	0.812	1	0.896	Fully Connected (Linear 35)
1	0.54	0.7012	0.685	1	0.813	Fully Connected (Linear 32)
1	0.62	0.765	0.725	1	0.841	conv. (Conv 5_3)
1	0.28	0.437	0.581	1	0.187	Conv. (Conv 4_3)
0.623	0.76	0.68	0.692	0.54	0.609	conv. (Conv 3_3)
1	0.56	0.717	0.694	1	0.8213	conv. (Conv 2_2)
0	0	0	0.5	1	0.167	conv. (Conv 1_2)

Figura 52: Resultados clasificador K-means para VGG16

5. Conclusiones y trabajo futuro

5.1. Conclusiones

La motivación de este trabajo surgió en parte por el conocimiento de un campo tan difuso y a la vez con tantas posibilidades de desarrollo en el futuro como son las redes neuronales. Ello, ha permitido poder aprender de muchos de los entornos y tecnologías vistas en el estado del arte y además aplicarlas de una forma práctica para resolver un problema real de procesamiento de imágenes.

Otro punto a destacar, ha sido el aprendizaje de un estilo de programación que deja a un lado la secuencialidad y aprovecha los recursos de la extensión Pytorch y las posibilidades que esto ofrece. Algo sin ninguna duda, imprescindible para el ahorro de recursos y la eficiencia en el código. Esta diferencia de programación de un modo vectorial y de un modo secuencial tiene inmensas diferencias.

En extracción de características de la última capa conectada, según nuestra implementación vectorial tiene un coste temporal mucho menor que la misma operación si se realiza de forma secuencial. Los tiempos de ejecución medios del extractor de características ronda los 15-20 segundos mientras que si se utiliza el secuencial este proceso puede aumentar significativamente hasta el punto de tardar horas en la ejecución del mismo para ciertas capas. Se observan muchos problemas de rendimiento difícilmente solucionables si se utilizan bucles anidados para recorrer las dimensiones de los tensores.

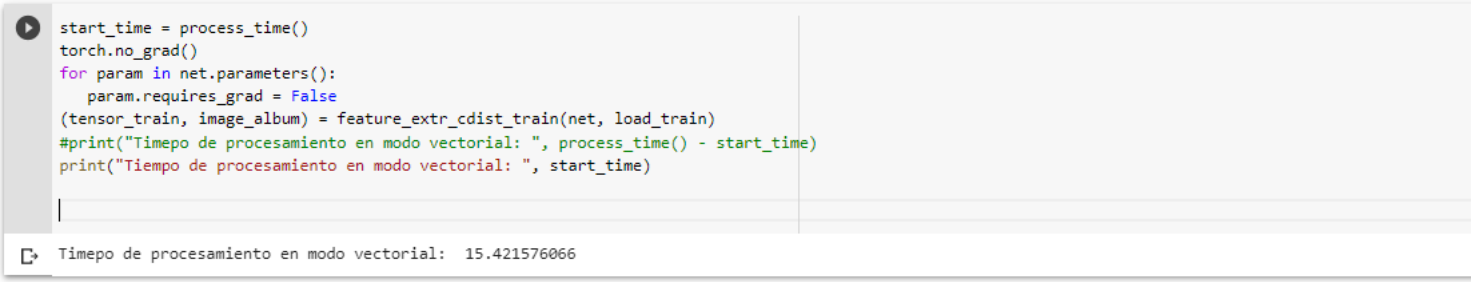


Figura 53: Tiempo ejecución modo vectorial

5.2. Trabajos futuros

Tras la realización de este trabajo han ido surgiendo a lo largo del desarrollo del mismo distintas posibilidades que se podrían llevar a la práctica y realizar nuevos estudios.

En primer lugar, una posibilidad sería utilizar otro tipo de redes convolucionales más avanzadas y de lanzamiento posterior a las ya mencionadas VGG16 y AlexNet.

Para un futuro trabajo, una posibilidad sería la realización de un estudio de granulidad fina en el que se aumente el número de clases, como podría ser un ejemplo en el dataset de interiores, clases que representen las diferentes habitaciones (salón, cocina, baño, dormitorio etc.).

Otra de las ideas que se han tenido presente ha sido realizar este trabajo utilizando datasets de la Universidad Autonoma y que bajo el mismo funcionamiento, se implemente un reconocedor basado en las características de la imagen y sea capaz de reconocer las distintas facultades que forman la universidad. De esta forma se conseguiría un reconocimiento de las escenas sin ningún tipo de etiqueta. De la misma forma, se podría implementar un algoritmo de clasificación que sea capaz de agrupar imágenes en función de las diferentes clases, que en este caso podrían ser cada una de las facultades.

Otro posible estudio sería el uso de las mismas redes que se han utilizado en este proyecto, AlexNet y VGG16 pero esta vez pre-entrenadas con las imágenes del dataset PLACES-365, especializado en reconocimiento de escenas exteriores, lo que permitiría mejor la precisión y sensibilidad, ya que ImageNet está más especializado en reconocimiento de objetos. Los resultados obtenidos para el dataset de exteriores previsiblemente será mucho mejor, aunque también sería interesante el comprobar el funcionamiento en un dataset de interiores para ver cómo se comporta.

Referencias

[1] Martin Armstrong. "The Future of A.I". En: (2016). URL: https://www.statista.com/chart/6810/the-future-of-ai/.

[2] Dave Gershgorn. "The Quartz guide to artificial intelligence: What is it, why is it important, and should we be afraid?" En: (2017). URL: https://qz.com/1046350/the-quartz-guide-to-artificial-intelligence-what-is-it-why-is-it-important-and-should-we-be-afraid/.

[3] Y. Bengio Y. LeCun y G. Hinton. *Deep learning*. 2015, págs. 436-444.

[4] B. K. Iwana S. Uchida S. Ide. "A further step to perfect accuracy by training cnn with larger data". En: *ICFHR* 14.3 (2016), págs. 405-410.

[5] A. Sharif Razavian H. Azizpour J. Sullivan y S. Carlsson. "Cnn features off-theshelf: an astounding baseline for recognition". En: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops* (2014), págs. 806-813.

[6] *¿Qué es Python?* URL: https://luca-d3.com/es/data-speaks/diccionario-tecnologico/python-lenguaje.

[7] *SAS. Líder en soluciones de analítica de clientes por Forrester*. URL: https://www.sas.com/es_ar/news/press-releases/locales/2018/sas-es-reconocida-como-lider-en-soluciones-de-analitica-de-clien.html.

[8] *What is Anaconda, Inc.?* URL: https://docs.anaconda.com/anaconda-cloud/faq/#what-is-anaconda-inc.

[9] *Conda Documentation*. URL: https://docs.conda.io/en/latest/.

[10] I. S. Ufimtsev y T. J. MartÃnez. *Graphical processing units for quantum chemistry*. Vol. 10. Computing in Science Engineering. 2008, págs. 26-34.

[11] *Running Python script on GPU*. URL: https://www.geeksforgeeks.org/running-python-script-on-gpu/.

[12] *CUDA developer information*. URL: https://developer.nvidia.com/cuda-zone.

[13] juan Antonio Gomar. *Qué son los Nvidia CUDA Cores y cuál es su importancia*. URL: https://www.profesionalreview.com/2018/10/09/que-son-nvidia-cuda-core/.

[14] Savia Lobo. *Is Facebook-backed PyTorch better than Google's TensorFlow?* 2017. URL: https://hub.packtpub.com/dl-wars-pytorch-vs-tensorflow/.

[15] *What is Pytorch?* URL: https://pytorch.org/tutorials/beginner/blitz/tensor_tutorial.html.

[16] *PyTorch Autograd. Understanding the heart of PyTorch's magic*. URL: https://towardsdatascien com/pytorch-autograd-understanding-the-heart-of-pytorchs-magic-2686cd94ec95.

[17] Jason Brownlee. *A Gentle Introduction to the Rectified Linear Unit (ReLU)*. URL: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.

[18] Stacey Ronaghan. *Deep Learning: Which Loss and Activation Functions should I use?* URL: https://towardsdatascience.com/deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56aa8.

[19] *Layer activation functions*. URL: https://keras.io/api/layers/activations/.

[20] Robert Hirsch. *Exploring Colour Photography: A Complete Guide*. Computing in Science Engineering. 2004.

[21] Andrea González Rodríguez. "Early detection of lung cancer through nodule characterization by deep learning". En: (2019).

[22] *Fully Connected Layers in Convolutional Neural Networks: The Complete Guide*. URL: https://missinglink.ai/guides/convolutional-neural-networks/fully-connected-layers-convolutional-neural-networks-complete-guide/.

[23] Jaime Durán. *Todo lo que Necesitas Saber sobre el Descenso del Gradiente Aplicado a Redes Neuronales*. URL: https://medium.com/metadatos/todo-lo-que-necesitas-saber-sobre-el-descenso-del-gradiente-aplicado-a-redes-neuronales-19bdbb706a78.

[24] vikashraj luhaniwal. *Forward propagation in neural networks — Simplified math and code version*. URL: https://towardsdatascience.com/forward-propagation-in-neural-networks-simplified-math-and-code-version-bbcfef6f9250.

[25] Varun Divakar y Rekhit Pachanekar. *Forward Propagation In Neural Networks*. URL: https://blog.quantinsti.com/forward-propagation-neural-networks/.

[26] Michael Nielsen. *Using neural nets to recognize handwritten digits*. URL: http://neuralnetworksan com/chap1.html.

[27] *Backpropagation In Convolutional Neural Networks*. URL: https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/.

[28] Diego Calvo. *Backpropagation – Redes neuronales*. URL: http://www.diegocalvo.es/backpropagation-redes-neuronales/.

[29] B. D. Hammel. *What learning rate should I use?* URL: http://www.bdhammel.com/learning-rates/.

[30] G. E. Hinton y R. R. Salakhutdinov. *Improving neural networks by preventing co-adaptation of feature detectors*. URL: https://arxiv.org/pdf/1207.0580.pdf.

[31] Dipanjan Sarkar. *A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in Deep Learning*. URL: https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a.

[32] *Transfer learning from pre-trained models*. URL: https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751.

[33] Sunita Nayak. *Understanding AlexNet*. URL: https://www.learnopencv.com/understanding-alexnet/.

[34] *AlexNet: The Architecture that Challenged CNNs*. URL: https://towardsdatascience.com/alexnet-the-architecture-that-challenged-cnns-e406d5297951.

[35] *¿Cómo funcionan las Convolutional Neural Networks? Visión por Ordenador*. URL: https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/.

[36] Alex Krizhevsky. *ImageNet Classification with Deep Convolutional Neural Networks*. URL: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[37] Hao Gao. *A Walk-through of AlexNet*. URL: https://medium.com/@smallfishbigsea/a-walk-through-of-alexnet-6cbd137a5637.

[38] Jerry Wei. *VGG Neural Networks: The Next Step After AlexNet*. URL: https://towardsdatascien com/vgg-neural-networks-the-next-step-after-alexnet-3f91fa9ffe2c.

[39] *What is WordNet?* URL: https://wordnet.princeton.edu/.

[40] Eman Ahmed y Mohamed Moustafa. "House price estimation from visual and textual features". En: *arXiv preprint arXiv:1609.08399* (2016).

Apéndice

En esta sección se mostrará el código implementado para las pruebas que se han realizado en este estudio. Se han añadido imágenes del extractor de características para el conjunto de datos de entrenamiento, el algoritmo de K-means y los reconocedores de escenas y lugares.

```
def feature_extr_cdist_train (model, dataloader):
    features_list = torch.empty( (0, 4096), dtype=torch.float ).cuda()                  #Tensor vacío de dimensiones la salida de la red (cambainte)
    image_album = torch.empty( [1, 3, 224, 224], dtype=torch.float ).cuda()             #Tensor de almacenado de las imágenes
    counter = 0                                                                         #Contador temporal
    model.cuda()                                                                        #Pasamos a cuda para usar GPU
    for i, data in enumerate(dataloader, 0):                                            #Bucle que recorre el mini-batch de imágenes

        input, label = data                                                             #Extraccion de Tensores
        input, label = input.to(device), label.to(device)                               #Se dividen datos en la entrada y etiquetas
        n,c ,h,w = input.size()                                                         #Se vuelve a pasar a cuda
        outputs = model(input)                                                          #Se sacan los tamaños de la entrada
        outputs = outputs.view(10, 4096)                                                #Se pasan las imagenes por nuestra red neuronal

                                                                                        #Bucle para la lectura de imágenes
                                                                                        #for j in range(10):
                                                                                        #    jeje = torch.transpose(input[j],0,1)
                                                                                        #   jejeje = torch.transpose(jeje, 1,2)
                                                                                        #  plt.imshow(jejeje.data.cpu().numpy())
                                                                                        # plt.show()

        if (i == 0):                                                                    #Condición inicial de diccionario vacío.
            features_list = torch.cat( (features_list, outputs[0].view(1,4096)), 0) #view(1,-1)   #Se añade la primera imagen
            image_album[0] = input[0]

        dist_tensores = torch.cdist(outputs, features_list, p=2.0)                      #Calculo distancias euclídeas
        activation = torch.gt(dist_tensores, AVG, out=torch.cuda.FloatTensor(len(outputs), len(features_list)))     #Comprobación con el umbral de decisión
        counter = len(features_list)                                                    #Calculo tamaño diccionario
        idx = activation.sum(1) == counter                                              #Condición de añadir si todos los tensores superan el umbral
        features_list = torch.cat((features_list, outputs[idx]), 0)                     #Concatenado del tensor
        image_album = torch.cat((image_album, input[idx,:,:,:]), 0)                     #Concatenado de la imagen

    print("Tamaño de Diccionario",len(features_list))
    return features_list, image_album
```

Figura 54: Extractor de características para conjunto entrenamiento programado vectorialmente

```
def feature_extr (model, dataloader):
    features_list = torch.cuda.FloatTensor().zero_()                    #Tensor Vacío
    counter = 0                                                         #Creamos un contador
    model.cuda()                                                        #Pasamos a GPU
    for i, data in enumerate(dataloader, 0):                            #Bucle para cargar las imágenes

        input, label = data                                             #Extraccion de Tensores
        input, label = input.to(device), label.to(device)               #Obtenemos clases y tensores de las imagenes
        n,c ,h,w = input.size()                                         #Pasamos a cuda
        outputs = model(input)                                          #n=batch size, c = numero de canales, h y w (dims imagen)
        feature = torch.cuda.FloatTensor(n,1000).zero_()                #Los tensores resultantes de las imagenes
        feature += outputs                                              #Creamos un tensor del tamaño del minibatch con el que se comparará la tabla de lugares
                                                                        #rellenamos el tensor con los tensores de la salida del modelo
        if (i == 0):                                                    #Si es la primera iteracion del batch se cargan 2 tensores en la tabla de referencia
            featuresss = torch.cuda.FloatTensor(2, 1000).zero_()        #Creamos un tensor de tamaño 2
            featuresss[0] +=outputs[0]                                  #Introduicmos 1º tensor
            featuresss[1]+= outputs[1]                                  #2º tensor
            features_list = torch.cat((features_list, featuresss), 0)   #Concatenamos en la lista de lugares


        for j in range(len(outputs)):                                   #Bucle que recorre hasta el nº de filas de la salida
            for k in range(len(features_list)):                         #Bucle que recorra la otra dimensión

                temp = feature[j]                                       #Asignacion temporal
                dist_tensores = torch.dist(features_list[k], temp, p=2.0)   #Se calcula la distancia del temporal con el del bucle interno
                if (dist_tensores > AVG):                               #Condición supera umbral
                    eature = torch.cuda.FloatTensor(2, 1000).zero_()    #Creacion tensor
                    eature[0] = temp                                    #Asignación del temporal
                    features_list = torch.cat((features_list, eature), 0)   #Se concatena
                    features_list=features_list[features_list.sum(dim=1)!=0]    #Suma de todas las dimensiones distintas de 0
    return features_list
```

Figura 55: Extractor de características para conjunto entrenamiento programado secuencialmente

```
def k_means_torch(dictionary, model):
    centroids = torch.randn(2, 4096).cuda()                          #Cremos centroides aleatorios
    dist_centroids = torch.cdist(dictionary,centroids, p=2.0)        #Distancia euclidea entre centroides y lugares del diccionario
    f = torch.sort(dist_centroids, dim=1)                            #Se ordenan estas distancias
    c1 = f.indices[:, 0].view(-1, 1)                                 #Redimensión para visualizar lso tensores como 1xN
    c2 = f.indices[:, 1].view(-1, 1)
    dicty1 = c1 * dictionary                                         #Se seleccionan solamente los más cercanos a ese centroide
    dicty2 = c2 * dictionary
    B1 = dicty1[dicty1.sum(dim=1) != 0]
    B2 = dicty2[dicty2.sum(dim=1) != 0]
    H1 = torch.mean(B1, dim=0).view(1,-1)                            #Se calcula la media de esos puntos asignados al centroides
    H2 = torch.mean(B2, dim=0).view(1,-1)
    cenrs = torch.cat ((H1, H2), 0)                                  #Se concatenan las medias de los puntos
    centers = (centroids + cenrs)/2                                  #Se actualizan los centroides
    while True:                                                      #Bucle hasta que se dejen de actualizar los centroides. Se repite el proceso comentado anteriormente

        dist_centroids_loop = torch.cdist(dictionary,centers, p=2.0)
        f_ = torch.sort(dist_centroids_loop, dim=1)
        c1_ = f_.indices[:, 0].view(-1, 1)
        c2_ = f_.indices[:, 1].view(-1, 1)
        dicty1_ = c1_ * dictionary
        dicty2_ = c2_ * dictionary
        B1_ = dicty1[dicty1_.sum(dim=1) != 0]
        B2_ = dicty2[dicty2_.sum(dim=1) != 0]
        H1_ = torch.mean(B1_, dim=0).view(1,-1)
        H2_ = torch.mean(B2_, dim=0).view(1,-1)
        cenrs_ = torch.cat ((H1_, H2_), 0)
        temp = (centers + cenrs)/2
        a = torch.all(torch.lt(torch.abs(torch.add(temp,-centers)), 1e-6))      #Umbral de 10e-6 ya que sino estaría iterando constantemente
        if (a == True):
          break
        else:
          centers = (centers + cenrs)/2
    return centers
```

Figura 56: Algoritmo K-means

```
def reconz_places (model, dataloader, dictionary, centers):
    model.cuda()                                                     #Pasamos a la GPU
    places = torch.empty((10, 0), dtype=torch.long).cuda()           #Se crea un tensor de tamaño 10 (para el mini-batch)
    for i, data in enumerate(dataloader, 0):                         #Bucle con las imágenes de entrada
        input, label = data                                          #lectura de datos
        input, label = input.to(device), label.to(device)            #Se pasa a GPU
        n,c ,h,w = input.size()                                      #Se sacan dims de los input

        outputs = model(input)                                       #La entrada pasa por nuestra red
        outputs = outputs.view(10, 4096)                             #Se cambia la forma en la que se visualiza el tensor de salida

                                                                     #Bucle para la lectura de imágenes

                 #for j in range(10):
                 #   jeje = torch.transpose(input[j],0,1)
                 #  jejeje = torch.transpose(jeje, 1,2)
                 # plt.imshow(jejeje.data.cpu().numpy())
                 #plt.show()
        dist = torch.cdist(outputs,dictionary, p=2)                  #Calculo distancia euclidea
        k = torch.sort(dist, dim=1)                                  #Se ordenan
        y = k.indices[:,0].view(1,-1)                                #cambio visualización
        if (i == 0):                                                 #Caso inicial
          places = y
        else:                                                        #caso habitual de concatenación
          places = torch.cat((places, y), 0)
    return places
```

Figura 57: Reconocedor de lugares para entorno de validación

```
def reconz_scenes (model, dataloader, dictionary, centers):
    model.cuda()                                                     #Pasamos a la GPU
    scenes = torch.empty((10, 0), dtype=torch.long).cuda()           #Se crea un tensor de tamaño 10 (para el mini-batch)

    for i, data in enumerate(dataloader, 0):                         #Bucle para las imagenes
        input, label = data                                          #Obtenemos clases y tensores de las imagenes


        input, label = input.to(device), label.to(device)            #Pasamos a cuda
        n,c ,h,w = input.size()

                                                                     #n=batch size, c = numero de canales, h y w (dims imagen)
        outputs = model(input)                                       #Los tensores resultantes de las imagenes
        outputs = outputs.view(10, 4096)                             #visualizacion de los tensores

        dist = torch.cdist(outputs,dictionary, p=2)                  #Calculo dist. euclidea
        k = torch.sort(dist, dim=1)                                  #Se ordenan los tensores
        y = k.indices[:,0].view(1,-1)                                #Utilzados solamente los que nos interesasn (el más cercano)

        dist_centroids = torch.cdist(outputs, centers)               #Distancia euclidea de los centroides y el diccionario
        g = torch.min(dist_centroids, dim=1)                         #Se obtiene el minimo
        if (i == 0):
          scenes = g.indices.view(1,-1)                              #Caso inicial
        else:
          scenes = torch.cat((scenes, g.indices.view(1,-1)), 0)      #Caso genérico
    return scenes
```

Figura 58: Reconocedor de escenas para entorno de validación